ScadaMobile

User Manual. Version 2.1.1

What is ScadaMobile.

SacadaMobile is a native iPhone and iPod Touch application for real time monitoring of industrial PLC based systems and processes.

In can be applied to building automation, industrial process control, mini hydro power plants, water plants, and anywhere where reliable, instant access to real time remote data is essential.

Main features.

- ✓ Local or remote access. Any number of concurrent PLCs.
- ✓ Very fast, independent of project size. immediate connection and display.
- ✓ Boolean, Integer and Floating Point values.
- ✓ Advanced Strings and Arrays Support
- ✓ Direct file import from Excel.
- ✓ Configurable Accounts and Access Levels.
- ✓ Alarms, Trend Graphs.
- ✓ Lookup Texts, Expressions.
- ✓ Direct connection to PLCs and RTUs without servers.
- ✓ TCP/IP based security.

How you can use ScadaMobile in 5 simple steps.

1. Create an Excel spreadsheet with the specs and behaviors of the Variables to be controlled. (See Data Source Files)
2. Export or Save into a CSV file.
3. Import the file created in step 2 into ScadaMobile (See Import)
4. Set the PLC IP address in ScadaMobile if you did not so in step 1 (See Network access).
5. Monitor process variable states and values from anywhere

TABLE OF CONTENTS

1 General Aspects 5
1.1 Supported protocols. 5

2 User Interface Elements 6
2.1 Tabbed interface 6
2.2 Variables in ScadaMobile 7
2.3 Home Tab Bar and Navigation Bar 8
2.4 Bottom Panel 9

3 Data Source Files 10
3.1 Data Sources created in Excel 11
3.1.1 Specification of Variable Names (column A) 12
3.1.1.1 Tag Scope 12
3.1.2 Specification of Variable Types (column B) 13
3.1.2.1 Representation of Character Strings in PLCs 15
3.1.3 Specification of Variable Addresses (column C) 17
3.1.3.1 Internal Tags 20
3.1.3.2 PLC Memory Arrays 21
3.1.4 Specification of Attributes (column D) 23
3.1.4.1 Attribute Scope and Kind. 24
3.1.4.2 Tag Attributes 25
3.1.4.3 Global Attributes 31
3.1.5 Pages, Sections, Rows and Data Sources 34
3.1.6 Look-up Tables 35
3.1.7 Alarms 36
3.1.8 Comments in Data Sources 37
3.1.9 Specification of Communication Protocol 37
3.1.10 International Languages Support and String Encodings 38
3.1.10.1 English and Western European Languages 38
3.1.10.2 String Encoding for International Languages. 38
3.1.10.3 Use of International Characters in PLC Strings 40
3.1.11 Expressions 41
3.1.11.1 Data types in Expressions 43
3.1.11.2 Supported Operators and Operator precedence 45
3.1.11.3 System Variables 46
3.1.11.4 Functions, Methods and more about Operators 47
3.1.11.5 The format specifiers and the format function 52
3.1.11.6 The if then else clause and the ternary conditional operator 53
3.1.11.7 Putting it all together. Advanced Expressions Examples 54

3.2 Omron's CX-Programmer as a Data Source generator. 56
3.2.1 Build a project in CX-Programmer. 56
3.2.2 Generate a Source File with variables from CX-programmer. 57
3.3 Rockwell RS-Logix 5000 as a Data Source generator 58
3.3.1 Build a Project in RS-Logix 5000. 58
3.3.2 Export Controller Tags from RS-Logix 5000. 58
3.4 Editing source files in a text editor. 59

4 File Import into ScadaMobile. 60
4.1 Source Files supported by ScadaMobile. 60
4.2 Other Files supported by ScadaMobile. 61
4.3 Custom Company Logo. 61

5 Connections. 62

6 User Accounts. 63

7 Network Settings for local access. 65
7.1 PLC Settings for local access. 66
7.2 ScadaMobile Settings for local PLC access. 67

8 Network Settings for remote access. 68

9 Security. 70
9.1 Validation Codes. 71

10 Background Task Processing 73

11 Performance 74

12 Examples 75

Document Revision History 85

1 General Aspects

ScadaMobile is presented in a tabbed interface. Every tab has its own role within the app and allows for different functions. With ScadaMobile you can monitor data coming from various PLCs.

Generally, you will use “Settings”, “File Server”, “Files” and “Connections” tabs during deploying stages.

The “Home” tab shows real time values of process variables in PLCs or calculated values. It is the one you will use for normal monitoring. ScadaMobile automatically goes to this tab on launch.

ScadaMobile uses the concept of “Data Sources” (see Data Sources) and “Connections” (see Connections) to do its job. “Sources” contain the definition of tags and variables, and “Connections” represent links with PLCs. A Source is always associated with a Connection, but a single Connection can belong to several Sources.

ScadaMobile also supports user accounts with access levels that can limit the ability to perform certain operations

1.1 Supported protocols.

ScadaMobile gets values from Industrial PLCs by polling them using TCP/IP industrial protocols. The following protocols are supported:

PROTOCOL NAME	SUPPORTED PLCs or Brands (Not exhaustive)	REMARKS
EIP/Native	Allen Bradley ControlLogix and CompactLogix	Native CIP communications using Ethernet/IP explicit messaging
EIP/PCCC	Allen Bradley SCL505 and Micrologix controllers, other controllers through 1761-NET-ENI	PCCC commands (DF1) encapsulated in Ethernet/IP.
FINS/TCP	Omron CS1, CJ1 and newest	For communication with Omron PLCs with ethernet communication capabilities.
Modbus/TCP	Schneider Electric, Automation Direct, Phoenix Contact, Wago...	For communication with PLCs and RTUs adopting the Modbus/TCP specification
Modbus over TCP	Serial Modbus RTU devices.	Can connect to Modbus devices through a simple Ethernet to Serial gateway.
Siemens/ISO_TCP	Siemens Simatic S7-1200, S7-300, S7-400 controllers	RFC 2126, ISO Transport Service on top of TCP for Siemens Step 7 programmable controllers.

2 User Interface Elements

This section provides an overview of the principal aspects of ScadaMobile user interface. It does not pretend to be exhaustive. Most interface elements will be described as needed in the sections that follow this one.

2.1 Tabbed interface

ScadaMobile uses the typical iPhone tabbed interface to organize several aspects of the app. On each tab a Navigation interface is usually presented. Not all tabs are available to all user accounts and not all the options in a tab are accessible to all users Five tabs are available (from left to right):

Settings.

Available to all users but with restrained options for non-Administrator users. You can create accounts, log to a particular account, set several user interface behaviors, and specify default communication settings.

Non Administrator users have all options banned except the Log In feature

File Server.

Only available to the Administrator user. Allows for managing, uploading and downloading files into ScadaMobile through its embedded Web Server.

Files.

Only available to the Administrator user. Presents a list of source files (containing tag definitions) and allows for placing a selection of the relevant ones for an application. You can also store and view any documents or reload the provided examples within this tab.

Connections

Available to all users but with restrained options for non-Administrator users. Presents a series of entries, called connections that represent actual links to PLCs. Within each connection you are able to check the communication status and to set validation codes for PLCs. Also, relevant communication settings and error information is presented for sources and connections. You can also switch monitoring on or off from this section.

Non Administrator users are only allowed to switch monitoring on or off from this tab. Connections and all related information is hidden for these users.

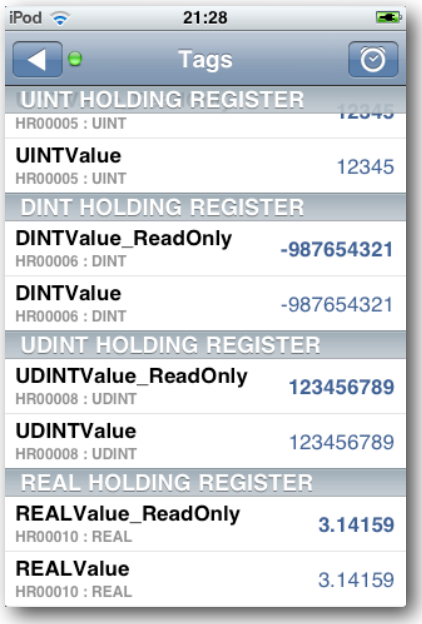
Home

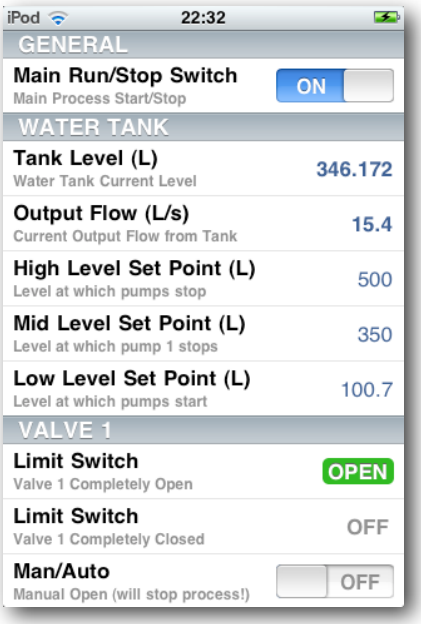
This is the main view and the place where actual tags, variable values and any calculations are presented. It is available to all users but access level on tags will be applied depending on the current source files configuration, so what each user views may vary. Trend graphs and Alarms are also displayed in this tab.

2.2 Variables in ScadaMobile

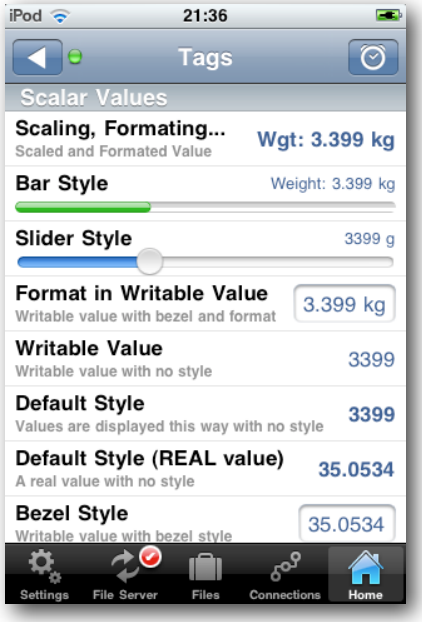
Process variables or Tag values coming from PLCs are organized in ScadaMobile as a list with sections in a similar way than the Contacts Application. Depending on variable type, style, and other characteristics, specific display may vary.

On the left side of each row, relevant information for identifying variables, such as their name, or their particular role in the monitored process is displayed. On the right side real time values of variables or suitable controls for interacting with them are shown.



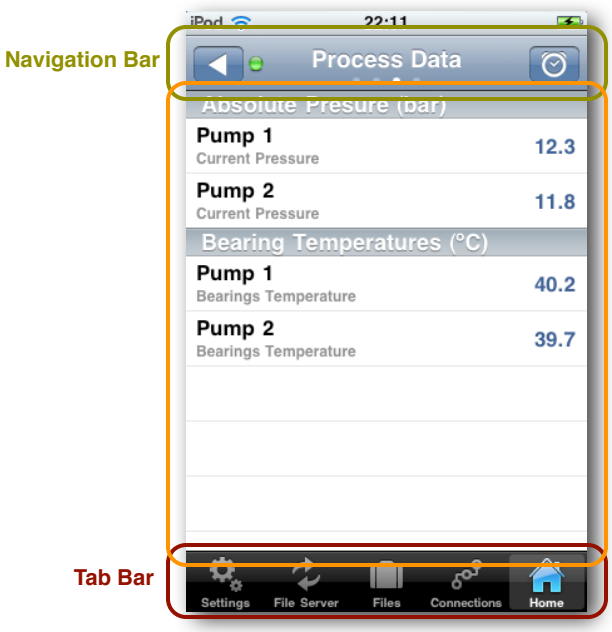


The provided examples demonstrate several ways for displaying elements on the Home view list.

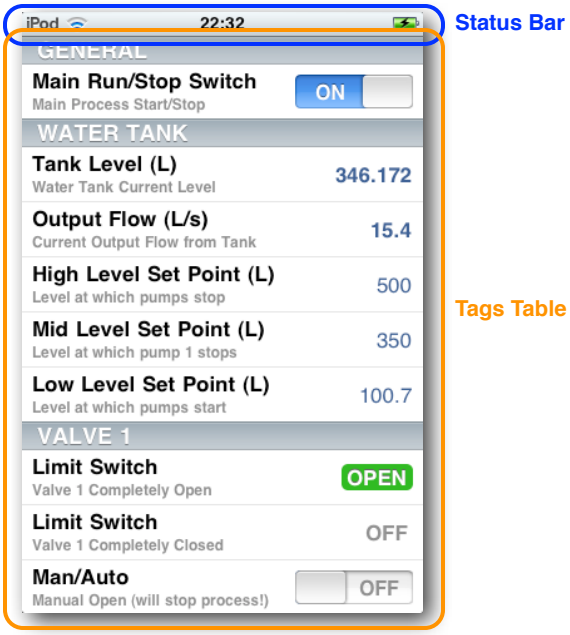


2.3 Home Tab Bar and Navigation Bar

The Home Tab is where variables are displayed and where users can interact with their PLC values. Like most iPhone productivity apps the interface consists on a Navigation Bar on top, and the Tab bar at the bottom of the screen. The standard iPhone status bar is always visible in ScadaMobile.



Tab Bar and Navigation Bar



Hidden Tab Bar and Navigation Bar

To optimize the available space on the Home Tab view, users can chose to hide the *Tab Bar* as well as the *Navigation Bar* as shown on the above screen shoots.

You can interact in the following ways on these basic interface elements.

- To hide or to show the *Tab Bar* on the Home View use the **Hide bottom bar** option on the *Settings* tab.
- To scroll to top of the *tags table* **tap on the status bar** of the iPhone screen.
- To hide or to show the *Navigation Bar* on the Home View use a **scroll down gesture** as you are **on top of the tags table**.
- To switch from one page to another use a **scroll left or scroll right gesture on the page title**. Alternatively tap on the right or left sides of the Page Control on the Navigation Bar (next to the page dots)
- To show or to hide the Page Control on the *Navigation Bar* use the **Paging Enabled** option on the *Settings* tab.
- To navigate to the list of pages tap on the *Navigation Bar* **left button**.
- To see alarms or trend graphs tap on the *Navigation Bar* **right button**.

2.4 Bottom Panel

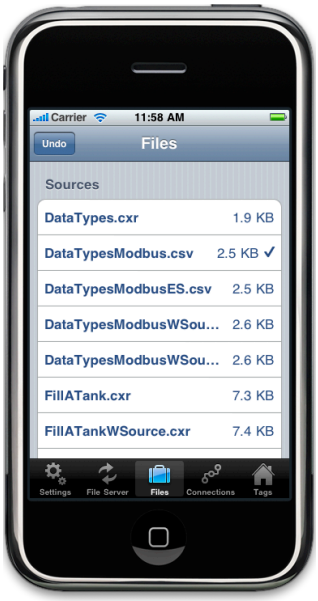
By tapping on the right button on the home view *Navigation Bar* the bottom panel will appear.

The various user interface elements in it will allow you to view and acknowledge alarms as well as create any number of trend graphs with any number of plots in them.

Just scroll the bottom panel left and right to move to different pages on it. The page control on the bottom also provides a way to switch among bottom panel pages

Bottom Panel

3 Data Source Files



Source Files contain information that ScadaMobile needs for visualizing PLC process variables as required by users.

ScadaMobile accepts csv files created in Excel, Open Office or a plain Text Editor. For some PLC brands, it partially supports as well files which are directly created from PLC vendor's development tools such as Omron's CX-Programmer, or Allen Bradley RSLogix.

Source Files imported into ScadaMobile are shown in the "Files" tab under the "Sources" section. One or several sources can be selected and variables from all of them will be presented depending on current user access level.

Each source refers to a single PLC, but several sources can refer to the same PLC. All sources that point to the same PLC are automatically joined into a single Connection.

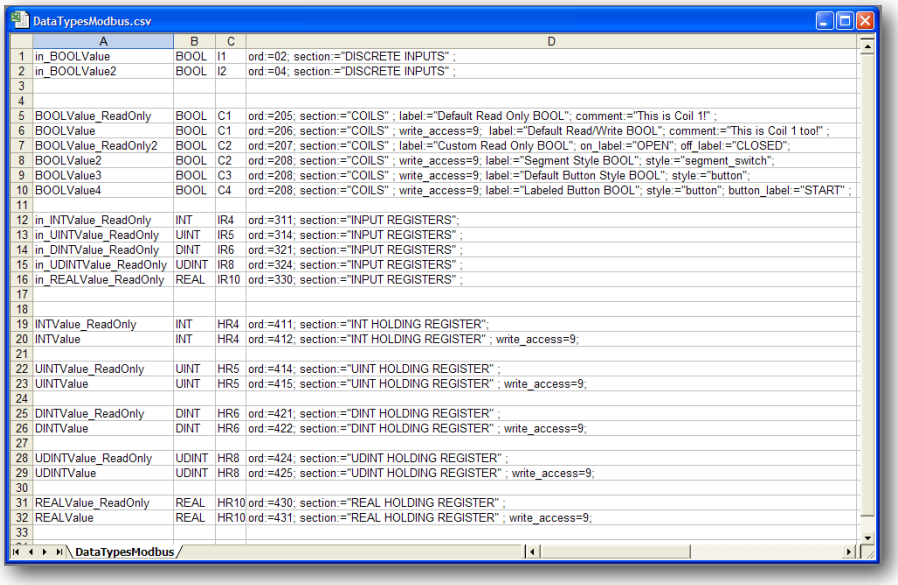
NOTE: Be sure you have at least one selected source or ScadaMobile will not display any variable or row on the Main View.

3.1 Data Sources created in Excel

You can use an Excel or Open Office to compose a data source file for ScadaMobile. The file must be exported or saved in CSV format.

> *NOTE: CSV file format is not identical for all language localizations due to different use of delimiters. ScadaMobile accepts any csv files in any of the supported languages, but unfortunately this is not the case for MS Excel. Consequently, Excel may not correctly open the provided examples on the wrong language.*

An Excel spreadsheet with the specification of variables consists of four (4) columns and one row per variable.


DataTypesModbus.csv

	A	B	C	D
1	in_BOOLValue	BOOL	I1	ord:=02; section:="DISCRETE INPUTS" ;
2	in_BOOLValue2	BOOL	I2	ord:=04; section:="DISCRETE INPUTS" ;
3				
4				
5	BOOLValue_ReadOnly	BOOL	C1	ord:=205; section:="COILS" ; label:="Default Read Only BOOL"; comment:="This is Coil 1!" ;
6	BOOLValue	BOOL	C1	ord:=206; section:="COILS" ; write_access=9; label:="Default Read/Write BOOL"; comment:="This is Coil 1 too!" ;
7	BOOLValue_ReadOnly2	BOOL	C2	ord:=207; section:="COILS" ; label:="Custom Read Only BOOL"; on_label:="OPEN"; off_label:="CLOSED";
8	BOOLValue2	BOOL	C2	ord:=208; section:="COILS" ; write_access=9; label:="Segment Style BOOL"; style:="segment_switch";
9	BOOLValue3	BOOL	C3	ord:=208; section:="COILS" ; write_access=9; label:="Default Button Style BOOL"; style:="button";
10	BOOLValue4	BOOL	C4	ord:=208; section:="COILS" ; write_access=9; label:="Labeled Button BOOL"; style:="button"; button_label:="START" ;
11				
12	in_INTValue_ReadOnly	INT	IR4	ord:=311; section:="INPUT REGISTERS";
13	in_UINTValue_ReadOnly	UINT	IR5	ord:=314; section:="INPUT REGISTERS" ;
14	in_DINTValue_ReadOnly	DINT	IR6	ord:=321; section:="INPUT REGISTERS" ;
15	in_UDINTValue_ReadOnly	UDINT	IR8	ord:=324; section:="INPUT REGISTERS" ;
16	in_REALValue_ReadOnly	REAL	IR10	ord:=330; section:="INPUT REGISTERS" ;
17				
18				
19	INTValue_ReadOnly	INT	HR4	ord:=411; section:="INT HOLDING REGISTER";
20	INTValue	INT	HR4	ord:=412; section:="INT HOLDING REGISTER" ; write_access=9;
21				
22	UINTValue_ReadOnly	UINT	HR5	ord:=414; section:="UINT HOLDING REGISTER" ;
23	UINTValue	UINT	HR5	ord:=415; section:="UINT HOLDING REGISTER" ; write_access=9;
24				
25	DINTValue_ReadOnly	DINT	HR6	ord:=421; section:="DINT HOLDING REGISTER" ;
26	DINTValue	DINT	HR6	ord:=422; section:="DINT HOLDING REGISTER" ; write_access=9;
27				
28	UDINTValue_ReadOnly	UDINT	HR8	ord:=424; section:="UDINT HOLDING REGISTER" ;
29	UDINTValue	UDINT	HR8	ord:=425; section:="UDINT HOLDING REGISTER" ; write_access=9;
30				
31	REALValue_ReadOnly	REAL	HR10	ord:=430; section:="REAL HOLDING REGISTER" ;
32	REALValue	REAL	HR10	ord:=431; section:="REAL HOLDING REGISTER" ; write_access=9;
33				

DataTypesModbus

ScadaMobile looks for the following information on each column

Column A : Variable Name.

Column B : Data Type.

Column C : Variable Address.

Column D : Tag or Global Attributes including Display and Communication Attributes

3.1.1 Specification of Variable Names (column A)

Variable names are entered in column A, they must *begin with a letter* and should *not contain spaces or special characters*. They can be just informative if they are not used elsewhere, or they can be functional if used as variables in expressions. Variables can hold *numeric* values, *strings* or *arrays* (see Expressions).

Variable names given in column A have a meaning within the ScadaMobile environment, but they are not necessarily aliases to actual Tag names in PLCs. PLC tags are referred by address or name (depending on protocol) and given in column C.

3.1.1.1 Tag Scope

Tags can be defined to have a **local** or **global** scope

Local variables are identified as per the general specification of variable names, that is beginning with a letter. Global variables are identified by placing a $ sign before their actual name.

Examples:

```
local_var
$global_var
```

Local variables have a scope limited to the data source file they are in. When a local name is found in an expression its definition i looked for only in the same file the expression appears, therefore you can use the same names and expressions in several source files without conflicts.

Global variables have an application wide scope so you can only have single instances of them across all selected source files. The advantage of using them is that you can access to their values from anywhere in your project even if the project is made of several source files.

3.1.2 Specification of Variable Types (column B)

Data types determine mostly how variables will be displayed. Non boolean Scalar Types can be represented on screen in various ways such as a number, a slider control, or a progress bar. Boolean values can be displayed in several formats depending on attributes.

Arrays of values can also be stored in variables and transferred from/to PLCs. By properly using array expressions you can retrieve individual values as desired. To indicate that a variable holds an array you append [n] to its data type. In such case, 'n' indicates the total number of elements that the array will hold.

The following types are supported.

DATA TYPE	REMARKS
BOOL*[n]*	Value that can adopt one of two states.
SINT*[n]*	8 bits signed integer value (-128 ... +127)
INT*[n]*	16 bits, signed integer value (-32768 ... +32767)
UINT*[n]*	16 bits unsigned integer value (0 ... 65535).
UINT_BCD*[n]*	4 digit BCD value stored in a 16 bit register using 4 bits per digit (0 ... 9999)
DINT*[n]*	32 bits signed integer value (-2147483648 ... +2147483647)
UDINT*[n]*	32 bits unsigned integer value (0 ... 4294967295)
UDINT_BCD*[n]*	8 digit BCD value stored in two 16 bit register using 4 bits per digit (0 ... 99999999)
REAL*[n]*	32 bits floating point value (IEEE 754) (aprox -1e38 ... +1e38)
CHANNEL*[n]*	Same as UINT
WORD*[n]*	Same as UINT
DWORD*[n]*	Same as UDINT
STRING*[n]* **STRING(size)*[n]***	Type containing a characters string. Actual representation depends on protocol, for example Allen Bradley controllers can hold up to 82 character bytes. Siemens S7 controllers require a *size* specification for strings. Notice that *size* is given between parentheses, NOT square brackets. Note that STRING[n] does not indicate a string containing *n* characters but an array holding *n* strings with default capacity. Particularly do not confuse with CHAR[n] or STRING(n) which will hold a single string with a capacity of *n* bytes. By default Strings on controllers are interpreted as per the WINDOWS-LATIN1 encoding, but other encodings are possible if a Explicit Encoding or a UTF-16 file is given (see International Characters Support). The STRING type should be used with the appropriate string memory area or string tag type in controllers supporting them. The use of the STRING data type is not limited to controllers with particular support for strings. See section **Representation of Character Strings in PLCs** below for further information.

DATA TYPE	REMARKS
CHAR[size]	Similar to STRING except that it does not insert a leading length word. It can be used on protocols with no specific support for strings such as Modbus. In this case *size* indicates the string buffer length, i.e. the number of character bytes that should be allocated in the PLC for the string, starting from the address specified in column C. Note that CHAR[20] would technically mean an array of 20 character bytes, however in this case it will be treated as a single string with a capacity of 20 bytes. Keep in mind that if you use a string encoding other that the default, you must expect the string to have a capacity of less than *size* characters. This is because on some encodings a single character may require multiple bytes to be represented. In the future we will make the size specification available for CHARs just like the STRING type. This will allow for having arrays of strings of char. For example CHAR(size)[n] will represent an array of *n* strings with a capacity of *size* bytes each. See also section **Representation of Character Strings in PLCs** below
LOOKUP	This is a special purpose type used to create a text entry on a table, which is referred to by variables having the 'lookup' style attribute. Rows with the LOOKUP data type on column B are just lookup data table entries and do not have any role on communications or add additional rows on screen. See also: Lookup Table
When specifying the tag type, you can optionally define an array size for it as shown above in italics. When you do so, the related variable will hold an array of values of the relevant type instead of a single value. See Memory Arrays for more information. Size definition is obligatory for CHAR types.	

3.1.2.1 Representation of Character Strings in PLCs

Strings in PLCs are stored in several ways depending on PLC brand or family. ScadaMobile uses a homogeneous way to indicate PLC tag types that in some cases differ slightly from the PLC manufacturer way. As a general rule, you do not generally need to worry about which particular representation a particular PLC uses. ScadaMobile handles it all automatically for you.

However, not all PLCs share the same fields for representing a string. For example Allen Bradley controller strings are fixed capacity and can hold up to 82 character bytes. Siemens S7 controllers, on the other hand, require a size specification for strings.

For Allen Bradley Controllers you can simply use STRING to indicate a single string, or for STRING[*n*] to indicate an array of *n* strings. The actual representation of a single string on the PLC consists on a UINT or UDINT field followed by a 82 bytes long buffer.

AB MICROLOGIX STRING REPRESENTATION (STRING):

Length (2 bytes)	Characters (fixed size, 82 bytes)

AB LOGIX STRING REPRESENTATION (STRING):

Length (4 bytes)	Characters (fixed size, 82 bytes)

For Siemens S7 controllers you must use STRING(*size*) where *size* is the total number of byte characters that the string can hold, or STRING(*size*)[*n*] to indicate an array of *n* strings of *size* character capacity. The actual representation of a STRING(size) in the PLC consists on the following pattern.

SIEMENS SIMATIC S7 STRING REPRESENTATION (STRING(size)):

Size (1 byte)	Length (1 byte)	Characters (variable size)

Although the above representations are the default ones for the mentioned controller brands, ScadaMobile will still chose one of them for use on any controller. Usually however you will want to use the raw char string representation CHAR[n]

RAW CHAR STRING REPRESENTATION (CHAR[size]):

Characters (variable size)

For raw char string reads, ScadaMobile will understand NULL character or the total buffer size as the termination of the string. On writes, ScadaMobile will pad all unused bytes with NULL characters. This is the usual convention for raw character string representations.

Important note about Strings with Siemens Simatic S7 controllers.

The size field for STRINGS in S7 must be generally specified. This is usually done in Siemens software by appending the size in square brackets just after 'STRING'. For example STRING[20].However, ScadaMobile already uses square brackets to identify arrays so this notation conflicts with S7 notation.

To work around this we chose to use normal parentheses to indicate size.

Thus, STRING sizes must be indicated in column B using round parentheses. The square notation is still reserved for arrays, so when you use them you will be referring to ARRAYs. Consider the following cases:

STRING(20) This refers to a STRING with a capacity of 20 characters and should not be confused by STRING[20]

STRING(20)[3] This is an ARRAY of 3 elements, each element is a string with a capacity of 20 characters

STRING[20] This is an ARRAY of 20 STRINGs. This is **not** a string of 20 characters!. Since the default string size for S7 is 254 (256 bytes including the size and length fields) you will end having an array of 20 STRINGs with a capacity of 254 characters each. Actually you will end reading (or writing) a range of 20*256 = 5120 bytes on your PLC for this tag and your PLC will most probably reply with an out of range error.

3.1.3 Specification of Variable Addresses (column C)

A Variable Address represents a memory location, a register or a Tag in a PLC to which a Variable refers. Addresses are specified in different ways depending on which communications protocol they belong to. Addresses belonging to different protocols can not be mixed in a particular Source File. You must use at least one separate Source File for each PLC and Communication Protocol.

The particular protocol to use can be specified by means of a comment on the first line as described in Specification of Communication Protocol. This comment may look something like this:

```
# %protocol eip/native
```

For protocols based on registers or memory locations, Variable Addresses are specified by a prefix referring to the appropriate memory area followed by a numeric value indicating the position in that area.Allen Bradley's Ethernet/IP for Logix Controllers is based on symbolic names. Write PLC Tag symbolic names in Column C

The following memory areas and prefixes are supported.

PROTOCOL	AREA PREFIX or TAG	REMARKS
EIP/Native (AB Logix Controllers)	***<symbolic-name>*** : Access by name	Actual symbolic PLC tag name. See Note on EIP/Native Communication Protocol below.
EIP/PCCC (AB Micrologix and SLC 5)	**O0**: Outputs **I1**: Inputs **S2**: Status **B3**: Binary **T4**: Timer **C5**: Counter **R6**: Control **N*n***: Integer File (*n* is file number) **F*n***: Floating Point File (*n* is file number) **ST*n***: String File (*n* is file number)	Tags are specified by File type, File number and Offset in the regular way. Individual bits in words can be accessed to using the usual slash notation for SCL and Micrologix controllers. Examples: B3:5 would access word 5 on file 3 of type 'B' N7:0 would access value at position 0 in N7 File. N7:0/3 would access bit 3 in N7:0
Fins/TCP (Omron)	**W**: Work area **D**: Data Memory Area (DM) **T**: Tim/Counter Area (T/C) **H**: Holding Register Area (HR) **A**: System Area (AR) Area **E**: Extra Memory (EM) Area (no prefix) : I/O Area	Individual bits are specified by following a dot (.) and a number from 0 to 15. For example: W10.5 refers to bit 5 of W10
Modbus/TCP Modbus over TCP	**I**: Input Discrete (read only) **C**: Coil **IR**: Input Register (read only) **HR**: Holding Register	To access Coil number 10, specify C10. To access Holding register 1 specify HR1. Individual bits in HRs can be accessed for reading or writing using a dot notation. For example, HR1.3 would refer to bit 3 in HR1

PROTOCOL	AREA PREFIX or TAG	REMARKS
Siemens/ISO_TCP (Siemens S7)	Area Prefixes: **E**: Inputs **I**: same as E **A**: Outputs **Q**: Same as A **M**: Internal Flags **DB*n*.DB**: Data block Valid Size Modifiers (after Area Prefix): **X**: Any size or 1 bit size **B**: byte (8 bits) **W**: word (16 bits) **D**: double word (32 bits) (none): 1 bit size	Tags are addressed by Area and Size in the usual way for S7 controllers. Examples: E2.3 accesses bit 3 of input address 2 I2.3 same as above (English notation) MB14 accesses address 14 on the flags area as a 8 bit value MW14 accesses address 14 on the flags area as a 16 bits value MD14 accesses address 14 on the flags area as a 32 bits value DB2.DBW6 accesses address 6 on Data Block number 2 area as a 16 bits value DB4.DBX8 accesses address 8 on Data Block number 4 area. Size depends on actual type specified type on column B

Accessing data types longer than one register.

For data types requiring more than one register or memory location, the lower address in their range must be specified. For example, a variable of type DINT addressed by HR100 will use HR100 and HR101 because 2 Modbus registers (16 bits) are required to accommodate the complete variable (32 bits). Integrators must be aware of it to avoid overlapping tag values. This applies to all protocols but EIP/Native.

Accessing a Register as a BOOL.

It is possible to specify a BOOL type for a register or memory location even if it is not meant to hold a BOOL. You can for example specify on a row that HR1 is a writable BOOL. In such case, ScadaMobile will use a switch control for that row and will write a value of one (1) or zero (0) to the register depending on user interaction on the switch.

EIP/Native Siemens/ISO_TCP does not allow a non BOOL PLC Tag to be treated as BOOL due to the strict type checking that this protocol encourages. You can use the ‘bool’ style instead to force the same effect.

Siemens/ISO_TCP enforces size identification along with memory area, thus some restrictions apply for use of BOOL type on larger sizes. In this case you can use the ‘bool’ style to force presentation to a BOOL type when necessary.

Accessing individual bits in a Register.

Individual bits on registers can be accessed by using the BOOL type and by specifying a bit address using the dot (.) or slash (/) notation depending on protocol (see table above). ScadaMobile will use the appropriate protocol command to avoid overwriting undesired bits on the register.

On EIP/Native you can still use the dot notation to access individual bits on variables, but due to strict type checking you must set the correct variable type on column B. In order to force ScadaMobile to display such values as bools you can use the 'bool' style.

Note on EIP/Native communication protocol (AB Logix controllers).

EIP/Native communications do not rely on particular memory locations or positions, but on symbolic names. With this protocol the user is relieved from the responsibility to assign memory addresses or registers and from the need to take tag sizes into account for storage. Additionally, EIP/Native tags carry data information such as type and size, which ScadaMobile uses to check against type mismatches on PLC returned values. As a result, it is not possible to store values that differ in type or size from the values that are uniquely defined in the PLC. Any attempt to so so will result in a 'type mismatch' error on the offending tag.

For **EIP/Native** symbolic names any valid reference to an existing scalar or array type tag including structure members or array elements is allowed.

For example "`myStructData[2,3].intMember`" may refer to an integer value referenced by the intMember member of element (2,3) of an array of structures. Look at 'EIP_TAG_Examples.csv' template for more examples of how to specify tag names for Allen Bradley Logix controllers.

As a general rule, any Tag name path referring to an existing scalar value (BOOL, SINT, INT, DINT, REAL, STRING) or array of such elements in a Logix Controller can be accessed by ScadaMobile.

To access arrays as a whole you need to define an array size next to the type, as discussed on the previous and following sections.

You can also access program tags by using the following syntax

`Program:<program_name>.<tag_name>`

Note that 'Program' is literal. *<program_name>* and *<tag_name>* identify just what they suggest.

Note also that ScadaMobile performs a Validation Code security check before any other attempt to access other tags is made, therefore, it is mandatory to have a tag named "SMValidationCode" of type INT in your PLC for communications to work. (see Validation Codes)

3.1.3.1 Internal Tags

Internal tags are stored and managed inside the ScadaMobile app. Internal tags mostly behave as actual PLC tags except that they are not linked to an actual PLC address. Thus, internal tags do not require an active TCP connection to display a value. In combination with expressions, internal tags are a powerful feature that allows for presenting calculated values to the user or holding intermediate values for subsequent use.

Internal tags support most of the available attributes except the ones specifically targeted at PLC tags such as the 'scale' attribute.

To specify that a Tag is internal use the word 'INTERNAL' instead of a PLC address or Symbolic Tag

PROTOCOL	AREA PREFIX or TAG	REMARKS
All	**INTERNAL** **LOCAL (deprecated)**	Indicates that this tag does not have a link to an actual PLC tag. Instead, it exists only in the app. Internal tags can be used to store and represent intermediate values, or expression results.

Just as regular PLC tags, internal tags can be associated to *Local* or *Global* scope variables. To identify a variable associated to an Internal tag to have a *Global* scope just prefix it with a $ sign.

3.1.3.2 PLC Memory Arrays

It is possible to read or write consecutive memory locations in the PLC as memory arrays and store them on a single variable. In order to do so you define the array size next to the specification of variable type (Column B). ScadaMobile will read the specified number of values and will store them in the relevant variable as an array. By subsequently using array expressions you are able to reach the individual array elements when required.

To deal with PLC arrays ScadaMobile uses several **access patterns** depending on specified type and actual size of data in the PLC. We will use examples based on the modbus protocol to discuss each possible case. The same patters will work on all protocols for similar types and data sizes. The following access patterns are possible:

1 - Accessing 1 bit data size memory areas as single values (modbus coils).

• BOOL, SINT, INT, DINT, in column B
• Cx in column C

The variable in column A gets the value of Cx (0 or 1) regardless of its type

Example: get value at C1 as INT

testTag INT C1
testTag will contain 0 or 1 depending on the value in *C1*.

2 - Accessing 1 bit data size memory areas as an array (modbus coils)

• BOOL[n], SINT[n], INT[n], DINT[n] in column B,
• Cx in column C

The variable in column A will be an array containing *n* elements of the specified type. Bits in each element will be taken from the PLC memory from the less significative to the most significative.

Example: get array of 2 INTs starting at C1

testTag INT[2] C1
Array element 0 (*testTag[0]*) will contain bits from *C1* to *C16*.
Array element 1 (*testTag[1]*) will contain bits from *C17* to *C32*.

Example: array of 10 BOOLs starting at C1

testTag BOOL[10] C1
Array element 0 (*testTag[0]*) will contain *C1*
Array element 1 (*testTag[1]*) will contain *C2*
...
Array element 9 (*testTag[9]*) will contain *C10*

3 - Accessing regular PLC memory as single values (valid on all protocols).

• BOOL, SINT, INT, DINT, REAL, STRING, CHAR[n] in column B
• HRx in column C

The variable in column A gets the value of HRx taking either the full register or the necessary following registers to hold the complete value. For types that are shorter than the actual register size the value in the PLC register is taken as a whole rather than trimmed to a shorter type.

Example: get DINT at HR1

testTag DINT HR1
testTag will contain the DINT value contained in *HR1,HR2*. (this is because DINT is 32 bits long and HRs hold 16 bits each)

Example: get HR1 as a BOOL

testTag BOOL HR1
testTag will contain 1 (true) if *HR1* is not zero, or 0 (false) otherwise. *HR1* raw value is therefore interpreted as boolean.

4 - Accessing regular PLC memory as an array of values (valid on all protocols).

• SINT[n], INT[n], DINT[n], REAL[n], STRING[n] in column B
• HRx in column C

The variable in column A will be an array containing *n* elements of the specified type. The array gets its values starting from HRx taking the necessary following registers to complete all its data according to data type size. Data is packed as it is found in PLC memory for types that are shorter than the actual PLC register size.

Example: get array of 2 REALs starting at HR1

testTag REAL[2] HR1
Array element 0 (*testTag[0]*) will contain the REAL value at *HR1,HR2*.
Array element 1 (*testTag[1]*) will contain the REAL value at *HR3,HR4*.

5 - Accessing individual bits of regular PLC memory (valid on all protocols).

• BOOL, SINT, INT, DINT, REAL in column B
• HRx.y in column C

The variable in column A gets the value of bit *y* (0 or 1) of *HRx* regardless of its type

For writes, using this pattern guarantees that writes of individual bits on registers will not affect or overlap other bits in the same or other registers.

Example: get HR1.0 as DINT

testTag DINT HR1.0
testTag will contain 0 or 1 depending on the value in *HR1.0*

6 - Accessing individual bits of PLC memory as an array of boolean values (valid on all protocols).

• BOOL[n], in column B
• HRx in column C

The variable in column A will be an array containing *n* elements of type BOOL. The array gets its values starting from Bit zero of HRx taking the necessary following registers to complete all its data.

Note that writing BOOL arrays with a size that is not a multiple of the raw register size on PLC memory will cause the exceeding bits to be set to zero.

Example: array of 32 BOOL starting at HR1

testTag BOOL[32] HR1
Array element 0 (*testTag[0]*) will contain bit 0 of *HR1*
Array element 1 (*testTag[1]*) will contain bit 1 of *HR1*.
...
Array element 16 (*testTag[16]*) will contain bit 0 of *HR2*
...
Array element 31 (*testTag[31]*) will contain bit 15 of *HR2*

Note on EIP/Native communication protocol (AB Logix controllers).

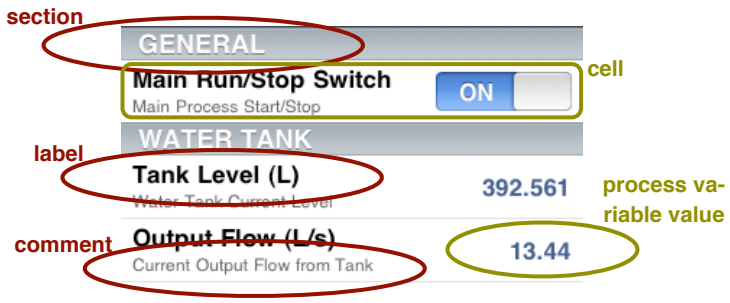
Since **EIP/Native** communications rely on symbolic names and type checking is performed on returned data, type matching must be observed. Basically, most of the above patters are applicable in the general way as far as the data type specified in column B matches the actual type on the PLC tag. This includes strings and arrays of any type.

From your perspective as integrator you do not need to treat Logix BOOL arrays in a special way as ScadaMobile handles them automatically for you, in essence you can either access individual elements by just entering BOOL on column B and the particular array element on column C (pattern 3) or you can get the complete (or part of the) array by specifying BOOL[n] on column B (pattern 6)

3.1.4 Specification of Attributes (column D)

Process Variables in ScadaMobile are represented in 'cells'.

The behavior and display of the monitored variables can be configured. In order to do so attributes can be set which determine that behavior. As an example, some of the available attributes are "section", "label" and "comment".



An attribute description in column D of a Source File follow this general pattern:

attribute = value;

For example, a boolean process variable like the one represented in the first cell on the figure above could have the following attribute description:

ord = 1 ; section = "GENERAL" ; label = "Main Run/Stop Switch" ;
comment = "Main Process Start/Stop" ; access = 3 ; write_access = 5;

The way the first cell on the figure is displayed implies that the current user has an access level of 5 or above because a selectable switch is available to change the process variable value instead of a static text (see User Accounts).

3.1.4.1 Attribute Scope and Kind.

Attributes by Scope.

Most attributes apply to a single Tag or Variable. They are referred to as **Tag Attributes**. Others have a global scope within a Source File, and we refer to them as **Global Attributes**. In the following sections all attributes are discussed individually.

Attributes by Kind.

Attributes can hold a **Numeric Value**, a **Text**, a **Special Text** or a **List of Values** depending on their meaning or purpose. For some attributes **Expressions** which calculate values depending on other variable values can be specified.

- **Numeric Values** are expressed as decimal numbers with optional decimal point and decimal digits if they have sense for the attribute.

  Several special values are provided for convenience. Particularly

  true : equals to 1.0
  false : equals to 0.0
  -inf : represents a very big negative number
  +inf : represents a very big positive number

  Numeric values are expressed directly after the equal sign without quotes, including the convenience values. Examples:

  write_access = 3 ;
  word_swap = true ;

- **Texts** usually represent text labels or fields in the application interface. They *must* be enclosed in quotes only if they contain spaces or the semicolon character, but it is advisable to do it always as a rule to maintain readability.

  label = "Main Run/Stop Switch" ;
  suffix = " %" ;

- Attributes requiring a **Special Text** only accept particularly chosen or formed text strings. Valid texts vary depending on the particular attribute.

  style = "bezel" ;
  format = “4.2” ;

- **Value lists** are used in cases where a single numeric value is not enough to provide the information required by the attribute. The general format is a list of numbers separated by commas and enclosed between ‘{‘ and ‘}’ characters.

  scale = { 0, 360, 0, 100 } ;
  bounds = { 0, 100 } ;
  color_bounds = { -inf, 50 } ;

- **Expressions** are accepted in some attributes to provide extended flexibility. Expressions are discussed on a separate section in this manual.

  hidden = my_tag == true ll my_other_tag == false ;
  value = flow1 + flow2 ;

3.1.4.2 Tag Attributes

A Tag Attribute is applied to a Tag or Variable at the same row and it exclusively affects to that Tag. All attributes are optional, and can be specified in any order.

The following attributes are supported:

TAG ATTRIBUTES	KIND	MEANING
page	text	Text indicating which page the variable belongs to. Text values must be written between quotation marks, for example: page="FIRST PAGE";
section	text	Text indicating which section the variable belongs to. Text values must be written between quotation marks, for example: section="SECTION ONE";
label	text	Main text in the cell representing a variable. If not specified, the variable name in column A will be used instead.
comment	text	Secondary text in the cell representing a variable. If not specified, a text comprising the variable address and its type will be displayed instead.
ord	number	Numeric value determining the order in which pages, sections and rows are displayed (see Pages, Sections and Rows below for a further discussion on this attribute)
access	number	Indicates the minimum access level an user needs to view a variable. By default **access=9;** is set.
write_access	number	Indicates the minimum access level an user needs in order to trigger changes (write to) a process variable value. Omitting this attribute forbids any change to the associated variable. Tags to witch the user is able to write are referred to as *Writable Tags*.
on_label	expression (string)	String Expression. Text displayed for a read only boolean variable when its value is 1 (true).
off_label	expression (string)	String Expression Text displayed for a read only boolean variable when its value is 0 (false).
button_label	expression (string)	String Expression In combination with the "button" style, it defines the text for a button.
format	special text	Text in the form "m.n" where *m* represents the minimum number of characters to be displayed. If the value to be displayed is shorter than this number, the result is padded with blank spaces (or zeroes if the number starts with zero). The value is not truncated even if the result is larger. *n* represents the number of digits to be displayed after the decimal point. An optional format specifier character can be appended to *n* for further customization. See Format Specifiers Example: **format="07.2";** will display the REAL value 12.345 as 0012.34 Example: **format="8x";** will display the DINT value -1 as ffffffff
prefix	text	Text to be prepended just before the variable value. Example: **prefix="$";**

TAG ATTRIBUTES	KIND	MEANING
suffix	text	Text to be appended to the variable value. Example: **suffix=" %";**
value	expression	Specifies a value to write on the related tag based on the result of an expression (see Expressions) Note that the 'value' attribute has a slightly different behavior for Internal tags than for regular PLC tags: * For PLC tags the result of the expression is first written to the PLC already converted to the relevant type and appropriately limited and descaled according to the 'bounds' and 'scale' attributes. On reading, the tag value is scaled back to the engineering unit. As a consequence a tag may end having a slightly different value after that. For example a Tag of type INT having scale={0,10,0,1}; will become 1.2 when assigning 1.234 to it through a 'value' change. Due to the scaling and type conversion process, the value in the PLC will be 12.. * On PLC writes involving STRINGS implicit conversions will also be performed if possible. For example, the STRING "2.3" will be converted to the number 2.3 upon writing a REAL tag. Similarly, the number 2.3 will result to the sequence of characters "2.3" when writing to a STRING tag. * For INTERNAL tags no type or scale conversion is performed on the result of the 'value' expressions. Internal tags are just given the result of the expression as it evaluates. Thus, the 'scale' and 'bounds' attributes will have no effect. Even the tag type is ignored to the effects of the assignment and the related variable may hold a completely different type. The following considerations also apply: * INTERNAL tags with a 'value' expression evaluating as a constant value will use it as the initial value for the tag. Otherwise, internal tags will be initialized to zero or an empty string upon initial project launch. * Writable INTERNAL tags will simulate a write-read round on a virtual PLC taking into account tag type, 'scale' and 'bounds' attributes with the same effects as a real write-read round as described above.
hidden	expression (boolean)	Numeric expression. Will make the row hidden when the result of the expression is non zero. Hiding all rows in a Section will remove the entire section including the section title. Hiding all rows/sections on a page will remove that page from the interface. Hiding/showing interface elements is fully animated and dynamic on evaluation of the 'hidden' expression. Dynamic hiding is useful to switch among several rows that selectively meet an arbitrary condition, or to force display parts of the interface depending on user or PLC triggered conditions.
style	special text	Attribute containing one of the texts below determining ways to represent or display the value of a variable.
		"switch" — It is the default style for writable booleans. It displays a boolean variable using a “Switch” control. The variable will change its state on each touch. Will be ignored if the variable is not writable.
		"segment_switch" — Displays an alternative to the “switch” style for writable booleans.

TAG ATTRIBUTES	KIND	MEANING	
		"button"	Presents a Push Button for writing of boolean values. Contrary to the “switch” style, the related process variable does not change permanently but it goes to 1 (true) on button press and goes to 0 (false) on button release.
		"bezel"	Draws a bezel line around a text field for highlighting writable numeric values. Note that it is only applicable to writable tags.
		“bool”	Presents the variable as a boolean even if it is not of type BOOL, limiting the possible values to 0 or 1. It also activates related attributes such as *on_label* and *off_label* and brings the special meanings of some attributes for bool types.
		"slider" "bar"	Presents a variable as a slider control or a bar depending on whether it is writable. Min and Max values are given with the *bounds* attribute. Both are identical and can be used indistinctly.
		"lookup"	Displays a text from a lookup table instead of the actual tag value. The tag value is used to determine which entry in the table is shown. This attribute will work both on read only tags and writable tags. On writable tags the bounds attribute is used to determine which range on the lookup table will be made available to the user for selection. See Lookup Table.
		"picker"	It is like the "lookup" style except that it will draw a bezel around the displayed lookup text when used on writable tags.
		"alarm"	Tags with this style are referred to as Alarm Tags and they are not shown on the main table but on the Alarms Panel. They can be combined with the *bounds* attribute. See Alarms.
		“barcode”	Activates the barcode reader for this tag. The tag must be writable (usually of type STRING). After reading a barcode with the device camera, the tag will be updated with the scanned code.
scale	value list	A four elements list in the form {x1,x2,y1,y2} where x1, x2 represent a pair of numeric values in raw units as present in the PLC and y1, y2 represent the same values in engineering units as will be displayed on ScadaMobile. By setting this tag attribute, raw values are converted (scaled) to engineering values on display by applying a linear transformation. This attribute can be specified in any read only or writable PLC tag. See also the 'value' attribute for a discussion on the particular support of this attribute on INTERNAL tags. Example **scale={0,100,0,1};**	

TAG ATTRIBUTES	KIND	MEANING
bounds	value list	A two elements list in the form {min,max} where min and max are numeric values used to indicate a range expressed in engineering units. This attribute can have several meanings depending on other attributes, particularly the *style* and the *write_access* attributes. On writable tags it determines and limits the available range of values that users will be able to enter On read/only tags its meaning depends on the particular style of the tag. It is currently supported by the "bar", "slider", and "alarm" styles. Example: **bounds={-100,100};**
color	special text	Indicates a color to apply to a variable value or control. Colors can be specified by name as listed in http://www.w3schools.com/css/css_colornames.asp. Colors can also be given by RGB value in hexadecimal format. The *color_bounds* attribute determines when the color will be applied. Supported values also include "TextDefault", "BarDefault" and "DefaultGreen" which are the colors used by default on texts, bars and boolean tags. Examples: **color = "red"; color = "olive" ; color = "#FF3300 ";**
color	expression (numeric)	Numeric Expression. The *color* attribute used in this way expects a 32 bit integer value containing the RGB color coordinates in the three lower significance bytes, with the B value in the least significant byte. The method *SM.color* can be used for convenience to generate a color from its RGB coordinates or name.
tint_color	special text	Indicates an alternate color to apply. The *color_bounds* and *tint_color_bounds* attributes determine color application ranges. Look at the *color* attribute for valid color values Example: **tint_color = "red";**
color_bounds	value list	A two elements list in the form {low,high} where low and high are numeric values determining the range of values for the tag where the color will NOT be applied, also known as an exclusion range. Examples: **color_bounds = {-inf,50};**. This will exclude tag values below 50 from displaying in the color set in the *color* attribute. Therefore tag values will be displayed in the specified color when they are above 50. **color_bounds = {-50,50};**. Tag values from -50 to 50 will be excluded from displaying in the color set in the *color* attribute. Therefore tag values will be displayed in the specified color only when they are below -50 or they are above +50, effectively enabling coloring for edge conditions. For boolean tags the *color_bounds* attribute has a predetermined value that will always make the tag appear in the *color* color for the On state.

TAG ATTRIBUTES	KIND	MEANING
tint_color_bounds	value list	A two elements list in the form {low,high} where low and high are numeric values determining the range of values for the tag where the tint color will NOT apply, also known as an exclusion range. Look at the *color_bounds* attribute for more information. When both *tint_color_bounds* and *color_bounds* attributes are specified, *color_bounds* takes preference over *tint_color_bounds*. However, *tint_color_bounds* still has the chance to act in the range that has been excluded by *color_bounds*, allowing for further color customization based on tag value. Combined Examples: **color_bounds = {-inf,50}; tint_color_bounds = {-inf,0};** Will display all values above 50 in the *color* color, positive values up to 50 will be displayed in the *tint_color* color. **color_bounds = {-50,50}; tint_color_bounds = {-20,20};** Will display all values below -50 and above 50 in the *color* color. Values from -20 to 20 will be excluded from the *tint_color* color, so the tint color will show for values from -50 to -20 and from +20 to +50. For boolean tags *color_bounds* and *tint_color_bounds* have predetermined values. You can simply use *tint_color* to display a color for both Off and On states, while *color* will supersede the color for the On state.
blink	expression (boolean)	Numeric Expression. Will cause any non writable tag to blink on the interface when the value assigned to the attribute is not zero (true)
blink_bounds	value list	A two elements list in the form {low,high} where low and high are numeric values determining the range of tag values where blinking will NOT be applied. If specified on a read only tag, the tag will visually blink with a period of 1 second for the values not in the specified range, when the expression assigned to the *blink* attribute is *true*. Example (to set a tag to blink always) : **blink_bounds = {-inf,-inf};** Example (to set a tag to blink when its value goes below -10 or above +10) : **blink_bounds = {-10,10};** Example (to set a boolean tag to blink for the On state) : **blink_bounds = {0,0};**
plot_color	special text	Determines the color to be used in trend graphs when plotting this tag. If omitted a sequential color is chosen automatically.

Tag Attributes for Modbus devices.

Modbus registers can be accessed on a PLC using serial communications through a Modbus RTU to Modbus/TCP Ethernet gateway. You can specify a slave id when using this feature.

MODBUS TAG ATTRIBUTES	KIND	MEANING
slave_id	number	This attribute identifies the associated *tag* as belonging to particular modbus device. Particularly, modbus serial devices with the specified id that are connected through a TCP gateway will be accessed. The default value is 1.
Note that the slave_id attribute for Modbus is a Tag attribute, so it applies only to the register it is next to. This is in contrast to the controller_slot attribute, which is global and applies to a source file.		

3.1.4.3 Global Attributes

Global Attributes are modifiers that apply to all tags or have a meaning in the context of the whole Source File. Global attributes can be placed anywhere in column D of source files. So they can be specified next to any variable and they will still have a global scope.

NOTE: If a particular global attribute is included more than once in a source file only its first occurrence will take effect

Communication settings are global and therefore they need to be specified only once in a source file.

COMMUNICATION ATTRIBUTES	KIND	MEANING
local_ip	text	Source address in text format for local access (LAN). Example: "192.168.1.40".
local_port	number	TCP port used for local connections (LAN) to this source. If it is not specified ScadaMobile will use the standard port for the protocol of the current Source File. (For example 502 for Modbus. Note that this can differ from the default port specified in ScadaMobile settings tab)
remote_host	text	Source address or symbolic DNS host name for remote connections. Example: "remote.remotehost.com".
remote_port	number	TCP port used for remote connections to this source (WAN-Internet). If it is not specified ScadaMobile will use the standard port for the protocol of the current Source File.
ssl	number (boolean)	Specifies whether TLS/SSL encryption is required for a remote connection. Non zero numeric values are considered true. Default value is 'false'.
validation_tag	special text	Allows for using a custom validation tag on protocols supporting it. • For EIP/NATIVE the validation tag name is always SMValidationTag, it can not be changed. • For EIP/PCCC use validation_tag = "N*x*:*y*"; only N files can be used and the code is stored as an INT (default is N98:0). • For MODBUS a validation tag is not supported. • For FINS/TCP use validation_tag = "D*x*"; only DM area can be used and the code is stored as a WORD (default is D19998). • For EIP/NATIVE the validation tag name is always SMValidationTag, it can not be changed. • For SIEMENS/ISO_TCP use validation_tag = "MW*x*"; only MW can be used and the code is stored as a WORD (default is MW998)
If no attribute from this group is specified, ScadaMobile will dynamically use default values or values from the application "Settings" tab view.		

Global Attributes for Modbus devices.

The Modbus specification does not exactly define how the data should be stored in registers or in which order the bytes or words are sent. The following global attributes help to deal with it. Swapped words/bytes options for modbus are global.

MODBUS GLOBAL ATTRIBUTES	KIND	MEANING
rtu_mode	number (boolean)	ScadaMobile will use "Modbus/RTU over TCP" instead of "Modbus/TCP". This will allow for accessing serial modbus/RTU devices behind an Ethernet-to-serial gateway not supporting MBAP. Use the 'slave_id' attibute on tags to route commands to the right modbus slave node.
word_swap	number (boolean)	Swaps words for 32 bit data (such as DINT or REAL) before sending to or upon receiving from a modbus device. Non zero values are ‘true’. Default value is ‘false’.
byte_swap	number (boolean)	Swaps bytes for 16 or 32 bit data before sending or upon receiving from a modbus device. Non zero values are ‘true’. Default value is ‘false’.
str_byte_swap	number (boolean)	Swaps bytes for string data before sending or upon receiving from a modbus device. Non zero values are ‘true’. Default value is ‘false’.

The combined effect swap attributes is as follows:

Assuming a default of 'ABCD' for byte order where 'A' is the Most Significative Byte (MSB) and 'D' is the the Less Significative Byte (LSB), you can combine 'word_swap' and 'byte_swap' with the following results:

1- 'word_swap=false; byte_swap=false;' will give 'ABCD' for 32 bit values and 'AB' for 16 bit values.
2- 'word_swap=false; byte_swap=true;' will give 'BADC' for 32 bit values and 'BA' for 16 bit values.
3- 'word_swap=true; byte_swap=false;' will give 'CDAB' for 32 bit values and 'AB' for 16 bit values.
4- 'word_swap=true; byte_swap=true;' will give 'DCBA'. for 32 bit values and 'BA' for 16 bit values.:

‘str_byte_swap’ is only attended in combination with the CHAR or STRING data type. It provides a way to swap odd and even bytes on character strings without affecting behavior for numeric data types.

NOTE: These attributes are only meant for Modbus communications and are ignored for the rest of supported protocols.

Global Attributes for Allen Bradley controllers.

Allen Bradley ControlLogix controllers can be plugged in any slot on the backplane. Ethernet/IP messages can be sent “connected” or “unconnected”. The following attributes can be used to determine these characteristics. These are global attributes.

EIP/NATIVE GLOBAL ATTRIBUTES	KIND	MEANING
controller_slot	number	Identifies the slot where the Logix controller is located. Default value is 0. It is ignored for EIP/PCCC communications (SLC and Micrologix)
connected_mode	number (boolean)	When true, ScadaMobile will use "connected messaging" instead of the default "unconnected messaging" for retrieving data from Ethernet/IP enabled PLCs. Look below for a discussion on what possible effects you might expect. Default value is ‘false’.

ScadaMobile currently supports two EIP mechanisms to send commands to AB PLCs:
(1) For a Micrologix or SLC it will send PCCC commands (DF1) embedded in EIP using a direct path.
(2) For a ControlLogix/CompactLogix it will send native CIP commands using a Backpane, Slot-Number path. The Backpane defaults to 1 and the Slot number is given in *controller_slot*.

ScadaMobile uses CIP Explicit Messages to retrieve and send data from/to Ethernet/IP enabled PLCs. Explicit messages can be sent "unconnected" or "connected". "Connected" messages require a Connection ID which is first asked to the PLC before sending other messages, while "unconnected" messages identify the specific path to the destination in the same message. Connected messaging is generally considered to be more reliable than unconnected because it reserves buffer space in the PLC for the message, and is therefore less likely to be blocked by other message traffic. However, if the TCP link between the message originator and the receiver is weak or prone to fail, unconnected messaging may be a better choice. Wireless spots or carrier networks can easily drop due to lack of coverage or weak signal, in these cases connected messaging communications may take longer to reestablish after a fault, resulting in less overall reliability and more user perceived delays than unconnected messaging. ScadaMobile uses unconnected messaging by default, but you can set it to use connected messaging for a source file by setting the *connected_mode* attribute to true.

Global Attributes for Siemens S7 controllers.

For Siemens S7 controllers you can use the ‘controller_slot’ to give appropriate ‘rack’ and ‘slot’ number

EIP/NATIVE GLOBAL ATTRIBUTES	KIND	MEANING
controller_slot	number	Identifies the rack and slot where the S7 controller is located. Bits 0-4 of this attribute value identify the slot number, while bits 5-7 identify the rack. Default value is 0.

3.1.5 Pages, Sections, Rows and Data Sources

Page, *Section* and *Ord* attributes

The ScadaMobile interface for the iPhone and iPod Touch consists on a set of cells or rows representing PLC variable values, which in turn are arranged in Pages containing Section headers for groups or rows.

Variables are optionally assigned a Section and a Page by attaching them the equally named attributes. All Variables having exactly the same Page name will belong and will be added to a page with such name. Similarly, the Section name attribute will be used to put all variables belonging to the same section under an epigraph with that name.

A single attribute, the 'ord' attribute, is used to force Pages, Sections and Rows to be placed in specific locations.

Pages and Sections are arranged according to their first appearance on the Source Files based on global order of all variables. This is used to force sections and pages to appear in the desired place. Variables without an order number will be considered to lay below or after ordered variables but they will maintain their relative position as found in the source file.

How it works

First, rows from *all* selected Data Source files are sorted from low to high using their 'ord' attribute. As said above, variables with no 'ord' attribute are moved to the end of the list always keeping their relative original order.

As sorting is performed, variables ord numbers are used to determine Pages and Sections position order.

Pages are arranged in position orders determined by the *smaller* ord number of all variables belonging to each page. In other words, a particular Page will be shown in the order given by the ord number of the *first* variable belonging to that page.

Within each Page, Sections are then arranged according to the *smaller* (or *first*) ord number of all variables belonging to that Section and Page.

Finally, variables are placed within their Page and Section according to their ord number.

Example

The following page/section/ord attributes in a series of tags from two sources will be arranged as shown below

Source 1

Tag	*Attributes*
TAG A	page = "Page 1"; section = "Section 1"; ord = 1 ;
TAG B	page = "Page 2"; section = "Section 1"; ord = 2 ;
TAG C	page = "Page 3"; section = "Section 1"; ord = 3 ;

Source 2

Tag	*Attributes*
TAG D	page = "Page 1"; section = "Section 2"; ord = 4 ;
TAG E	page = "Page 2"; section = "Section 2"; ord = 5 ;
TAG F	page = "Page 3"; section = "Section 2"; ord = 6 ;

Resulting arrangement of tags in Pages and Sections

Page 1
Section 1
TAG A
Section 2
TAG D

Page 2
Section 1
TAG B
Section 2
TAG E

Page 3
Section 1
TAG C
Section 2
TAG F

3.1.6 Look-up Tables

The Look-up table feature allows a variable to pick up a text string from a previously entered list based on current variable value and display said text instead of the tag value.

To have a Row display a look-up text instead of the usual value, use the 'style' attribute with "lookup" or "picker" as value

```
style = "lookup" ;
style = "picker" ;
```

If you want to show the lockup text alone with no label and centered as shown in the StylesExampleModb example, you can set the tag label to an empty text

```
label = "" ;
```

The actual table is given in additional rows in the CSV file by using special tags of type LOOKUP (in column B) and by specifying the related indexes in column C. The text written on column D or a comment attribute on the same row will be the Lookup Text.

The following table is an example of what you would need to write for creating a two entry table with indexes 1 and 2.

Column A	Column B	Column C	Column D
entry1	LOOKUP	1	This is look-up table at index 1
entry2	LOOKUP	2	This is the second entry in the table so it will show if a lookup tag value is 2

Lookup indexes on Column C do not need to be ordered or contiguous, they can be any number that fits in 16 bits (0 to 65535).

Multiple user lookup tables

You can specify an *access* level for entries in column D. In such case you must set the lookup text in the *comment* attribute. This allows for having different texts depending on user level. For example consider the following:

Column A	Column B	Column C	Column D
entry1_boss	LOOKUP	1	access=9; comment="Only me will see this message. I'm the boss!"
entry1_worker	LOOKUP	1	access=3; comment="I will see this with my user level of 3 or more";

Multiple range lookup tables

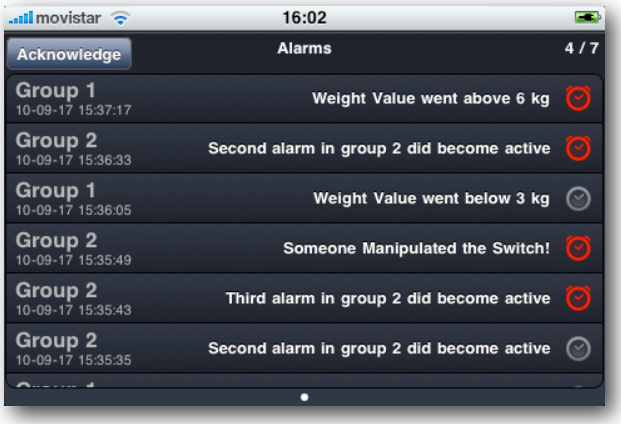
Lookup styled variables use the engineering unit as the index to the table. Therefore you can use the scale attribute for PLC tags to make a lookup styled row to access to particular portions or ranges of the table.

For example scale={0,10,100,110}; will forward any raw value coming from a PLC tag in the range from 0 to 10 to table text entries from 100 to 110.

You can use this feature to effectively have several virtual tables simultaneously in use. The StylesExampleModb example shows how this concept works.

Note, however, that since INTERNAL tags do not perform any value scaling the above technique will not work on them. Instead you can use appropriate numerical expressions in the 'value' attribute in order to fetch the desired range from the table. Additionally, the global lookup table is available to expressions with the `SM.lookup` method.

3.1.7 Alarms



An Alarm tag is a special kind of tag that is not displayed in the main tags table. An alarm tag is characterized because its *style* attribute is set to "alarm". Alarm conditions can be specified with the *style* attribute in combination with the *bounds* attribute attached to the tag.

Alarms are tracked and listed on a separate table as shown in the landscape screenshot next to this section. When an alarm condition is triggered, the tag *label* attribute is used to display the text on the left, which may refer to an alarm group, and the *comment* attribute contains the alarm message displayed on the right.

Alarms will remain on the list as long as they remain active or they have not been acknowledged. Their current state will be shown by small icons next to the alarm text:

- Bright Red Alarm Clock icon means active and not acknowledged
- Dark Red Icon means active and acknowledged
- Gray Clock Icon means inactive and not acknowledged

For a scalar type variables such as INT or REAL, the bounds attribute determines the exclusion range of the alarm condition. In other words, an alarm styled tag will be considered active if its current value is NOT in the specified range. For example consider the following:

style="alarm"; bounds={0,100};

The tag holding these attributes will be treated as an alarm, which will become active when its value is below 0 or above 100.

To support simple min or max condition alarms, the special numeric values "-inf" and "inf" can be used (see Attribute Scope and Kind, for a discussion on these values). For example

style="alarm"; bounds={-inf,100};

will make the alarm to trigger only when the value on the tag is above 100.

The default value for the *bounds* attribute on alarm tags is **bounds={0, 0};**. Therefore, any non-zero value in a tag will trigger an alarm. This is specially relevant for BOOL tag alarms, because by default they will become active when the tag goes to 1 (true) and inactive otherwise, just as you would expect.

Example

The following rows on a source file represent the settings for the alarm tags that generated first and forth rows on the screenshot in this page:

```
Alarm1   INT   HR1  label="Group 1"; comment="Weight Value went above 6 kg"; style=alarm; bounds={-inf,6};
Alarm4   BOOL  C1   label="Group 2"; comment="Someone Manipulated the Switch!"; style=alarm;
```

Performance Considerations

Unless regular tags, alarm tags are continuously polled from PLCs even if they are not shown on the screen. Also, ScadaMobile may continue polling them while running in the background. So special care should be taken when deciding what tags will be reserved for alarms. Particularly, it is advisable to group alarms in tags as contiguous as possible, and to use boolean alarms over scalar value alarms for as much as possible. In protocols supporting arrays of BOOL, they will be the best choice. Observing this recommendation will lead to shorter communication patterns and less network overhead than if no special care was taken, ultimately improving the end user experience.

3.1.8 Comments in Data Sources

You can comment Rows on Source Files for documental purposes or while you are testing your project. To do so just start the row with the '#' character. Examples:

```
# These are the lookup table entries for the style selection texts we are using
# This alarm checks for a value being too high on the statistics page
```

3.1.9 Specification of Communication Protocol

Starting with version 1.5 you can explicitly set the communication protocol on source files, instead of having ScadMobile determining it by the kind of addresses used in Column C as before. This feature has been provided to grant future compatibility with a larger number of communication protocols, where some addressing naming conventions could conflict or overlap existing ones.

To tell the actual communication protocol on a source file you must insert the following comment as the first line.

```
# %protocol <protocol_string>
```

As <protocol_string> you can use one of the following;

```
eip/native
eip/pccc
fins/tcp
modbus/tcp
siemens/iso_tcp
```

For example to communicate with an Allen Bradley ControlLogix controller you would have the following as the first line on your source file:

```
# %protocol eip/native
```

> *NOTE: Protocol Specification in this way will be made obligatory in the future for all protocols, so it is recommended for integrators and advanced users to use it on new projects and to set a plan to edit existing source files to conform with it.*
>
> *Since version 2.1 you must explicitly indicate protocols eip/native and siemens/iso_tcp.*

3.1.10 International Languages Support and String Encodings

The ScadaMobile iOS app includes localizations for English and Spanish for its user interface interface. However, International Characters and Strings in any language are fully supported. Integrators can therefore chose to present their project interface in any language.

To represent strings the concept of *String Encodings* is used. String Encodings are international conventions that determine how characters representing particular languages are stored into files and device memory.

3.1.10.1 English and Western European Languages

By default ScadaMobile assumes source files and Strings to conform to the **Windows Latin1 encoding**. This is adequate for English and most Western European languages such as German, French, Spanish, Portuguese, and many others. It does not require any particular setting. English and Western European versions of Excel on Windows also use the Latin1 encoding for CSV files. On Apple Mac computers you must save your CSVs as 'csv-windows' format to conform to this encoding.

The Latin1 encoding is backward compatible with old plain ASCII, meaning that ASCII characters share the same codes when represented in Latin1 encoding.

3.1.10.2 String Encoding for International Languages.

If you use a language that can not be represented with the Latin1 encoding, you are still able to import your files containing international characters in text attributes. Additionally, international characters are supported in Strings and they can be written and read back from PLCs without breaking their particular encoding.

There are two ways for using international characters in ScadaMobile:

Unicode UTF-16 encoding (16 bit multibyte encoding)

The UTF-16 encoding is capable for representing any character from virtually any language in the wold. To make use of this encoding you can do the following.

- Create a source file with texts in your language using your localized version of Excel as usual.
- Export the file as Unicode UTF-16 text.
- Import the file as usual in ScadaMobile. ScadaMobile will automatically detect files encoded as UTF-16. Unicode strings will be converted to UTF-8 before storing them on PLC memory.

> *NOTE: Strings in source files containing UTF-16 characters will be converted automatically to UTF-8 regardless of the Explicit Encoding specified on the source file as defined in the next sub heading. Any Explicit string Encoding given in a source file will be thus ignored if the file was already UTF-16 encoded.*

Explicit Localized encodings (8 bit or multibyte encodings)

Instead of using UTF-16 you may chose to use one of the 8 bit or multibyte encodings that are commonly used by default on localized versions of Windows and Excel. To do so follow the next steps:

- Create a source file with Strings on your language using your default localized version of Excel as usual.
- You must explicitly tell ScadaMobile the right string encoding of your file. See below.
- Export the file as usual. Excel will use the default encoding for your localization.
- Import the file in ScadaMobile. ScadaMobile will use the explicit encoding to open the file.

Setting a Explicit String Encoding

To set a explicit *String Encoding* for your file place the following command embedded in a comment on top of your file

```
# %encoding <string_encoding>
```

.

For example to set Japanese you can write

```
# %encoding Japanese/Win
```

The following explicit *string encodings* are supported:

EXPLICIT ENCODING	Description
WindowsLatin1	Identifies the ISO Latin 1 encoding (ISO 8859-1). This is the default.
UTF-8	Identifies the Unicode UTF 8 encoding.
MacRoman	Identifies the Mac Roman encoding. Used on western localizations of Mac OS. Useful when you use diacritic characters (Spanish, French, German, the degree º simbol...) but you do not want to export your file as csv-windows.
Cyrillic/Mac	Identifies the Mac Cyrillic encoding
Cyrillic/Win	Identifies the Windows Code page 1251 Slavic Cyrillic encoding
Cyrillic/ISO	Identifies the ISO 8859-5 Cyrillic encoding
Japanese/Mac	Identifies the Mac Japanese encoding
Japanese/Win	Identifies the Windows Code page 932 Japanese encoding
Japanese/JIS	Identifies the Shift-JIS format encoding of JIS X0213
Chinese/Mac	Identifies the Mac Simplified Chinese encoding
Chinese/Win	Identifies the Windows Simplified Chinese encoding
Chinese/GB2312	Identifies the GB_2312 Chinese encoding

Strings in source files containing UTF-16 characters will be converted automatically to UTF-8 regardless of the encoding specified on the source file.

The UTF-8 encoding is a multibyte character encoding derived from UTF-16. Like UTF-16 it can represent every character of all languages, but unlike UTF-16, it is backward compatible with ASCII, using only one byte for representing ASCII characters.

3.1.10.3 Use of International Characters in PLC Strings

You can store international Strings in PLCs with ScadaMobile just as easily as you do with ASCII strings. ScadaMobile will use the source file encoding identification to decode/encode strings onto raw bytes in the PLC.

When storing international Strings into PLCs you must expect the number of bytes used, and thus the PLC string length, to be larger than the number of characters the string actually holds. This is particularly notorious when storing Chinese or Japanese strings in PLCs.

The UTF-8 encoding, for instance, can use up to 6 bytes per character in a PLC. However, this does not affect how strings are allocated in ScadaMobile or the behavior of String methods and operators in expressions, since these always refer to actual characters and actual character lengths regardless of encoding.

Of couse, if you only use English or ASCII characters with an encoding that is backward compatible with ASCII, or you use the Latin1 encoding, only one byte per character will be allocated in your PLC to store strings.

3.1.11 Expressions

Expressions are allowed on several attributes and provide an advanced way to customize various aspects of the interface and behavior of ScadaMobile projects.

Expressions allow for combining variables with *operators* and *methods* to produce custom results. Variables are referenced in expressions by using their names as entered in **Column A**. There are also several system variables that can be used for special uses.

Event Driven Architecture.

Expressions in ScadaMobile are stored in a compiled form and are executed by an event-driven engine. The execution engine keeps expression reference information in a way that value changes trigger expression evaluation. The engine is not endlessly executing 'for' loops but only *change events*.

The expressions engine keeps references to referring expressions to create a tree like network where all expressions have links to other expressions. When a PLC tag changes, or an user interacts on a control, a *change event* is triggered. This event, that occurs at some point in the expressions network, is propagated through the relevant links to reach only the expressions that need to know about it, generally only a few.

The result is that expressions execution time is basically independent of project sizes or the total number of expressions defined in projects. The Event Driven Architecture is specially suitable for running ScadaMobile in the constrained environment of a mobile device and still be able to support a very large number of tags with no noticeable performance penalties.

Using Expressions.

ScadaMobile expression syntax is based on the open source Ruby scripting language syntax. For basic operations this syntax is similar to that of the 'C' programming language and virtually identical to all modern scripting languages.

The Ruby language was chosen because it features a clean, easy to learn, object-oriented syntax with a particular focus on expressions allowing for practical ways to represent and dealt with several data types and formats with great flexibility. ScadaMobile supports most operators including all common Logical, Arithmetic and Comparison operators, as well as commonly used Ruby functions and methods.

Support of Ruby expressions in ScadaMobile is a subset of the Ruby language. Expressions are not, and do not pretend to be a complete implementation of Ruby. In some cases we provided a single way to accomplish something that can be done in several ways on Ruby, and in other cases we integrated several functionalities in single methods instead of implementing all of them. So it is important to refer to this manual if you are also using a Ruby tutorial to determine what it is actually supported on ScadaMobile and which behavior differences may apply.

For those who already used Ruby, one of the most obvious differences between 'pure' Ruby and ScadaMobile is treatment of boolean values. Ruby treats everything as objects, including numbers, while ScadaMobile keeps the traditional 'C' like behavior. For example, in Ruby any number used in a boolean expression is a *true* value even if it holds zero. ScadaMobile, on the other hand, will still interpret 0 (zero) as *false* and non-zero as *true*, in the traditional sense of earlier programming languages, and hopefully in accordance to what PLC programmers would expect or feel more natural.

NOTE: If you are not familiarized with Ruby and want to test some ScadaMobile expressions before you start using them in your project you can try them on any of the available Ruby interpreters to help you understand how they work.

- *On any Mac OS X or Linux computer open the Terminal and type **irb**. This will launch the interactive Ruby interpreter. On the >> prompt type or paste any expression you would like to try.*
- *Ruby can be installed on Windows as well. Refer to this to begin with: http://www.ruby-lang.org/en/downloads/.*
- *You can even type Ruby expressions right on your Internet Browser!. Go here: http://TryRuby.org/*

Just type or paste the expression examples in this manual on any of the mentioned tools and see whether they give the expected results.

Values used in expressions or assigned to tag attributes using expressions are represented in engineering units. ScadaMobile do not have a notion of PLC raw values except for writing or reading values from PLCs. This is important in order to understand why assigning a 'value' to an INTERNAL tag may not give the same result than writing it to a PLC tag and reading it back. Differences are due to the implicit scaling/descaling and type conversion performed on PLC tags. See the 'value' attribute description for a discussion on this subject.

When using expressions in your project you must know the following:

* **Tag names in expressions are case sensitive**. This means that a tag named `int_value` will not be the same as a tag named `Int_value`.
* **Logical or Comparison operators assume non-zero values to be *true* and zero values to be *false***. The result of a Logical or Comparison operator, however, is always a value of 0 or 1.
* **Comparison operators are non-associative**. This means that expressions such as `value=a<b<c;` are not valid. You must use `value=a<b && b<c;` instead.
* **Assignments in expressions are not supported** except for assigning values to attributes. Therefore expressions such as `value = condition && (int_tag = 3 );` will cause a syntax error on the second assignment operator. Do not confuse the *assignment* operator = with the the *equality* operator == which *is* fully supported.
* **An expression is executed only when at least one of the referred variables or tags *change***. The process is totally transparent and integrators might not need to know about how it works underneath. However, keeping the event driven nature of ScadaMobile in mind can help integrators to understand why and when PLC tags will be written or alarms will trigger as a consequence of an user interaction or PLC Tag change that originated a cascade of change events.
* **Expressions containing Logical operators are no exception to the event driven design**. They will be fully executed even if a change occurs on the right side of the Logical operator. For example the expression `value = condition1 && condition2;` will be always *false* if `condition1` is *false*, however it will execute anyway as a consequence of a change on `condition2`. Although the expression result will not change (it will remain *false*), the engine will still send a *change event* to any referring expressions, which could potentially cause other effects such as a PLC tag rewrite if the expression was linked to a PLC tag.
* **Expressions are not allowed to create circular or recursive references**. This means that a result of an expression can not be refitted to another expression that ultimately would send a *change event* to the originating expression. This is not allowed at any level on the expressions execution chain. For example the following row `int_value INT INTERNAL "value=int_value+1;"` is not valid because the `int_value` tag creates a circular reference around the 'value' expression.

3.1.11.1 Data types in Expressions

Variables and Expressions in ScadaMobile support three basic data types, **Numbers**, **Strings** and **Arrays**. Appropriate *operators* and *methods* allow for conversion among types and to perform custom operations with great flexibility. See the following sections for a discussion on methods and operators.

Mixing Numbers, Strings or Arrays is only possible through the use of the appropriate *operators* and *methods* that result in compatible types. An direct consequence is, for instance, that you can not concatenate a number to a string unless you convert the number to a string first. Also, some operators have particular semantics depending on type just as most modern scripting languages including Ruby do. See next sections for a further discussion on this an other subjects.

Numeric values.

Numeric values in expressions and variables are internally stored as Double Float values (64 bits) Values read from PLCs including BOOL and INT tags are scaled and converted to Double Float type before they are used in expressions. All Arithmetic, Logical and Comparison operations are performed as Double Float operations. You may never expect to obtain truncated values from arithmetic calculations.

The above statement may change in the future to give support for true integer arithmetic. Currently, an implicit conversion to an integer type is only performed for bit or bitwise operations on numbers, and indexed access to string or array elements. In other cases you can use the to_i method to explicitly get the integral part of a numeric value according to your needs.

Constant numbers can be represented with optional decimal point and a base 10 exponent. Additionally, hexadecimal and binary notations are supported by using the *0x* or *0b* prefixes. The special forms *true*, *false* *+inf* and *-inf* are supported as well.

Examples:

-1.42 (decimal representation)
1.1666e+2 (decimal representation with exponent)
0xe0af (hexadecimal representation)
0b011011101 (binary representation)
true (same as 1)
false (same as 0)
-inf (very big negative number)
+inf (very big positive number)

Strings.

Strings are arbitrary sequences of characters that are manipulated as a whole. Strings can be read from or written to PLCs,. Several operations can be performed on strings such as concatenate, split or substring extraction by using the appropriate *operators* or *methods*. String literals are represented enclosed in double quotes. Strings are internally encoded in a compatible type (usually UTF8) but several encodings are supported for reading/writting to PLCs.

Examples :

```
"This is a literal string"
"Дискретные датчики"
"ピーエルシーのアラーム表示"
```

Arrays.

Arrays are-indexed collections of objects. Each element in an array is associated with and referred to by an index.

Array indexing starts at 0. A negative index is assumed relative to the end of the array, that is, an index of -1 indicates the last element of the array, -2 is the next to last element in the array, and so on.

Arrays can hold any objects such as Numbers, Strings or other Arrays. Arrays can be read from or written to PLC or can be created on demand in expressions by just enclosing their elements in square brackets.

Example:

```
["element at index 0", 123.4, [ 33, temperature]]
```

The above expression represents an array of three elements.

At index 0 we have a literal string: "element at index 0".
At index 1 we have a numeric value: 123,4.
At index 2 we have an array of 2 elements with the number 33 and the variable ‘temperature’ as their components

Elements of the referred array can be accessed by index as shown next:

`["element at index 0", 123.4, [ 33, temperature]][1]` would return `123.4`

`["element at index 0", 123.4, [ 33, temperature]][-3]` would return `"element at index 0"`

`["element at index 0", 123.4, [ 33, temperature]][-1][0]` would return `33`

`["element at index 0", 123.4, [ 33, temperature]][2][1]` would return the value of *temperature*

3.1.11.2 Supported Operators and Operator precedence

The following table shows the available operators and its precedence. The table lists all operators from highest precedence to lowest.

OPERATOR	Description	Associativity
()	Parentheses (grouping).	from inner to outer
. () []	Method selection, Method/Function call, Array or String subscript	left-to-right
! ~ + -	Logical NOT, Bit Complement, Unary plus, Unary minus.	left-to-right
* / %	Multiply, Divide, Modulo	left-to-right
+ -	Addition/concatenation, Subtraction	left-to-right
&	Bitwise AND	left-to-right
^ |	Bitwise XOR, Bitwise OR	left-to-right
< <= > >= != ==	Comparison operators	not associative
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
?:	Ternary conditional operator	right-to-left
if then else	Selective if then else clause	right-to-left

Operators are used in the usual way as per the Ruby or “C” language. Depending on data types involved the same operator may have a different meaning. See Methods, Expressions and more about Operators.for further information

3.1.11.3 System Variables

There are a number of system variables that can be used in expressions as if they would be regular tags. The following variables are provided.

VARIABLE NAME	TYPE	Description
$SMPulse10s $SMPulse30s $SMPulse60s	number (read only)	They generate a square wave signal with the period implicit on the variable name. They can be used to implement a Keep-Alive tag, to write periodically a value on a PLC, or to trigger periodic events for any purpose.
$SMAckButton	number (read only)	Variable linked to the Acknowledge button on the alarms panel. Goes to 1 when the button is pushed and 0 when it is released.
$SMCommState	number (read only)	A value indicating the current communication state of ScadaMobile. Possible values are the following: 0 - Communications running with all PLC connections linked. 1 - Monitor is switched off. 2 - One or more PLC are not linked or a new connection is in course. Partial link state. 3 - General communications error. No communication is established. This variable can be used to implement alarms related to PLC reachability or to show/hide interface elements depending on PLC availability.
$SMCurrentPage-Name	string (read/ write)	Name of the currently displayed page
$SMDate	string (read only)	Text representation of the current date and time in the following format: "yyyy-MM-dd HH:mm:ss"

3.1.11.4 Functions, Methods and more about Operators

Methods can be applied to intermediate expressions or variables to perform type conversions or to achieve particular requirements. They are like computer language functions that perform particular tasks. Not all *methods* are applicable to all types and their meaning can vary depending on type. *Methods* are invoked by appending a dot (*method selector operator*) followed by its name to the variable or subexpression they apply to.

O*perators* can also have a different meaning depending on the type of data they are applied to.

In the following tables we represent the meaning of the applicable operators and methods depending on data type.

Numeric operators and methods.

NUMERIC	Description
`num` *operator* `num2`	Arithmetic, comparison, logical operators produce the expected results. Available operators are listed on the operators precedence table shown earlier. The bitwise and complement operators extract the integral part of the operands before computing the result Example: `2+2` returns `4`
`num[n]`	Returns bit *n* from the integral part of *num*. Bit 0 is the least significant bit. The result can be only 0 or 1. Example: `3[0]` returns `1` Example: `3[1]` returns `1` Example: `3[2]` returns `0`
`num.to_i`	Returns the integral part of *num*. Example: `3.666.to_i` returns `3`
`num.to_f`	Returns *num*
`num.to_s` `num.to_s(fmt)`	Returns a string representation of *num* optionally formatted according to *fmt*. For a description of possible format specifiers refer to the *format* function. Example: `3.666.to_s("%02d")` results in `"03"` Example: `3.666.to_s("%02f")` results in `"04"` Example: `3.666.to_s` results in `"3.666"` Example: `25.to_s("%02.1f °C")` results in `"25.0 °C"` (Note that specifying a format in *to_s* is not a standard feature of Ruby)
`num.chr`	Returns a string containing a single character represented by the Unicode character code *num*. Example: `72.chr` would return `"H"`
`num.abs`	Returns the absolute value of *num*. Example: `(-3.66).abs` would return `3.66`
`num.round`	Returns *num* rounded to the nearest integer. Example: `3.66.round` would return `4`

String operators and methods.

STRING	Description
"*characters*"	Creates and returns a string containing the sequence of characters written between quotes.
`str[n]`	Gets the Unicode representation of the character at index *n* in *str*. If *n* is negative indexes start at the last character. Generates an error when attempting an out of bounds access. Example: `"Hello world"[0]` returns 72 (72 is the Unicode character representation of 'H') Unicode representation of English Language characters fully match the 7 bit standard ASCII character representation.
`str[n,m]`	Substring. Returns a substring of *str* starting at *n* and continuing for *m* elements. Always returns a string. Returns an empty string *""* on out of bounds access, *m* can not be negative. Example: `"Hello world"[0,4]` results in `"hello"` Example: `"Hello world"[-5,5]` results in "`world`" Example: `"Hello world"[6,5]` results in "`world`"
`str+other_str`	Concatenation. Example `"hello" + "world"` will give `"hello world"`
`str1` *comparison_operator* `str2`	String Comparison. Returns 1 or 0 (true or false) when comparing two strings for equality or as if they were sorted in a dictionary. Example `"alpha"<"beta"` returns *true* because "beta" is after "alpha" in a dictionary.
`str.to_i`	Parses a *str* into an integer value or returns 0 if not possible Example: `"3".to_i` returns 3
`str.to_f`	Parses a *str* into a floating point number or returns 0 if the conversion is not possible Example: `"3.2".to_f` returns 3.2
`str.to_s` `str.to_s(fmt)`	Returns *str*. formatted according to *fmt* if specified, or *str* otherwise Only the "s" format specifier is relevant for strings. Example: `"World".to_s("Hello %s")` would give `" Hello World "`
`str.split(str2)` `str.split`	Creates an array of strings by splitting *str* using *str2* as a delimiter but not including it. If *str2* is an empty string it splits *str* into each one of its characters. If *str2* is not given it returns *str* embedded in an array. Example `"08-04-2014".split("-")` returns `["08","04","2014"]` Example `"08-04-2014".split("")` returns `["0","8","-","0","4","-","2","0","1","4"]` Example `"08-04-2014".split` returns `["08-04-2014"]`
`str.length`	Returns the number of characters in *str* Example `"Hello".length` returns 5

Array operators and methods.

ARRAY	Description
`[d1,d2,...]`	Embeds the elements *d1*, *d2* and so on in an array and returns it. Array elements can be numbers, strings or other arrays. Example: the following expression `[1,"two",[10,"eleven"]]` would create an array containing three elements having a number at position 0, a string at position 1 and a two elements array at position 2.
`arr[n]`	Get element at index *n* from *arr*. If 'n' is negative indexes start at the last character. Generates an error when attempting an on out of bounds access Example: `["one","two","three","four"][1]` returns `"one"` Example: `["one","two",”three","four"][-1]` returns `"four"`
`arr[n,m]`	Subarray. Returns a subarray of *arr* starting at *n* and continuing for *m* elements. Always returns an array. Will return an empty array `[]` on access beyond limits. *m* can not be negative. Example: `["one","two","three","four"][0,2]` returns `["one","two"]` Example: `["one","two","three","four"][-3,1]` returns `["two"]` Example: `["one","two","three","four"][2,2]` returns `["three","four"]`
`arr + arr1`	Returns a new array built by concatenating the two arrays together Example: `["one","two"]+["three","four"]` returns `["one","two","three","four"]`
`arr.join(str)`	Returns a string created by converting each element of *arr* to a string, using *str* as a separator Example: `["one","two",3,4].join(":")` returns `"one:two:3:4"`
`arr.fetch(n,d)`	Tries to return the element at position *n* or returns *d* if *n* lies outside the array bounds. *n* must be numeric value representing the element index. Negative values of *n* count from the end of the array. *d* can be any numeric, string or array value. Example: `["one","two","three","four"].fetch(0,"none")` returns `"one"` Example: `["one","two","three","four"].fetch(4,"none")` returns `"none"`
`arr.length`	Returns the number of elements in *arr*. Example: `["one","two"].length` results in `2`

Built-in functions.

FUNCTIONS	Description

`format(fmt,...)`	Returns a string in which the list of arguments following *fmt* is formatted according to *fmt*, *fmt* is a Formating specification string. Formatting specifications in *fmt* are essentially the same as those for the sprintf function in the C programming language. Conversion specifiers in *fmt* begin with % and are replaced by a formatted string of the corresponding argument. A % character followed by another % will yield the '%' character. A list of supported conversion fields is given on the next section.
	Examples:
	`format("Room Temperature is: %4.1f", 25)` will return the string `"Room Temperature is: 25.0"`
	`format("%02d:%02d:%02d",hours,minutes,seconds)` may return the string `"01:15:48"` assuming the variables 'hours' 'minutes' and 'seconds' contain the same values.
	`format("Throughput: %4.1f%%", 25)` will return the string `"Throughput: 25.0%"`

System-methods.

SYSTEM	Description
`SM.lookup(n)`	The *lookup* method provides expressions access to the global Lookup table (see section Look-up Table for a discussion on lookup tables) This method picks the text from the global table at index *n* and returns it as a string. If the table does not have an entry for *n* returns `"--"` Example: Provided that entry 1 of the lookup table is '`Entry at index 1`' then: `SM.lookup(1)` will return the string `"Entry at index 1"`
`SM.color(r,g,b)` `SM.color(str)`	Returns a numeric representation of a color ready for use on the *color* attribute. You can either provide the RGB color coordinates as values ranging from 0 to 255 in *r*, *g* and *b*, or you can provide a string with the color name in *str.* Refer to the *color* attribute for further information on available colors.

Math-methods.

MATH	Description

`Math.atan2(y,x)`	Computes the principal value of the arc tangent of *y*/*x*, using the signs of both arguments to determine the quadrant of the return value.
	The *atan2()* function is used mostly to convert from rectangular (*x*,*y*) to polar (*r*,*θ*) coordinates that will satisfy *x = r*Math.cos(θ)* and *y = r*Math.sin(θ)*.
	In general, conversions to polar coordinates are computed in this way:
	`r = Math.sqrt(x*x+y*y)`
	`θ = Math.atan2(y,x)`

`Math.cos(x)`	Computes the cosine of *x* (measured in radians)
`Math.exp(x)`	Calculates an exponential function (e raised to the power of x)
`Math.log(x)`	Calculates the natural logarithm of x.
`Math.log10(x)`	Calculates the base 10 logarithm of x.
`Math.sin(x)`	Computes the sine of *x* (measured in radians)
`Math.sqrt(x)`	Computes the non-negative square root of *x*
`Math.tan(x)`	Computes the tangent of *x* (measured in radians)
`Math.PI`	Returns the π constant number

3.1.11.5 The format specifiers and the *format* function

The built-in function *format* returns a string formatted according to a format string like the usual printf conventions of the C language. In addition, *format* accepts *%b* for binary. ScadaMobile. format specifiers adopt the following form:

%<flags><width><.precision>specifier

Where *specifier* is the most significant one and defines the type and the interpretation of the value of the corresponding argument ('<' and '>' denote optional fields).

Note that the string provided for the *format tag attribute* or the format string passed to the *to_s* method have a slightly different purpose and may have less available options.

The following format conversion specifiers are available:

FORMAT SPECIFIER	Description	*format* attribute support	*format* function support	*to_s* method support
b	Binary integer	YES	YES	YES
c	Single character	NO	YES	YES
d,i	Decimal integer	YES	YES	YES
e	Exponential notation (e.g., 2.44e6)	YES	YES	YES
E	Exponential notation (e.g., 2.44E6)	YES	YES	YES
f	Floating-point number (e.g., 2.44)	YES	YES	YES
g	Use the shorter of e or f	YES	YES	YES
G	Use the shorter of E or f	YES	YES	YES
o	Octal integer	NO	YES	YES
s	String or any object converted using *to_s*	NO	YES	YES
u	Unsigned decimal integer	YES	YES	YES
x	Hexadecimal integer (e.g., 39ff)	YES	YES	YES
X	Hexadecimal integer (e.g., 39FF)	YES	YES	YES

For the meaning and possible contents of the optional *flags*, *width*, and *precision* fields refer to the sprintf specification:

http://www.cplusplus.com/reference/clibrary/cstdio/sprintf/

Note that since there is no need for it the *length* field is not available neither in Ruby or ScadaMobile.

3.1.11.6 The *if then else* clause and the ternary conditional operator

Two forms of conditional execution are provided:

(1) The ternary conditional operator provide conditional execution of expressions. Its syntax is the following:

```
expr ? expr1 : expr2
```

Returns *expr1* if *expr* is not zero (true) or *expr2* otherwise.

(2) The *if then else* clause provide conditional choice of expressions. It is used as follows:

```
if expr [then] expr1 [else expr2] [end]
```

Executes *expr1* if *expr* is not zero (true). If *expr* is zero (false) *expr2* is executed instead. Items between brakets are optional.

The *if then else* clause (2) differs from the ternary conditional operator (1) in the following ways:

- (1) executes when any of *expr*, *expr1*, *expr2* generate a change event. The result is always updated and will be consistent with the values of *expr*, *expr1* and *expr2* at all times. The execution will in turn trigger relevant change events up the expressions tree just as any expression would do.
- (2) only attends to *expr* change events. Any changes in *expr1* or *expr2* will not have an effect until *expr* executes. Furthermore, if *expr* is *false* and *expr2* was not specified, the execution tree is trimmed at this stage and no further execution up the expresion tree will happen.

(2) is useful in cases where you want to achieve a differential effect, for example to trigger an event when a condition goes from *false* to *true* but not the oposite. This is not possible with (1) because it will always execute both ways. Consider the following:

```
start   BOOL   INTERNAL   label = "Start Button"; style = "Button"; write_access=0;
stop    BOOL   INTERNAL   label = "Stop Button"; style = "Button"; write_access=0;
motor   BOOL   C1         label = "Motor State"; value = if start then 1 else (if stop then 0);
```

The previous lines will display a Start tand a Stop button. When the Start button is touched 1 will be written to C1. When the stop button is touched 0 will be written to C1.

Another use of the *if then else* clause is the implementation of a counter:

```
reset       BOOL   INTERNAL   label = "Reset Button"; style = "Button"; write_access=0;
increment   BOOL   INTERNAL   label = "Tick Button"; style = "Button"; write_access=0;
counter     DINT   HR1        label = "Counter"; value = if reset then 0 else (if increment then counter+1);
```

The Reset and Tick buttons will provide in this case the interface for a counter value written to the PLC.

Now consider the following case

```
color1          UDINT   INTERNAL   label = "Color 1"; write_access=0;
color2          UDINT   INTERNAL   label = "Color 2"; write_access=0;
colorSelection  BOOL    INTERNAL   label = "Select Color"; style = "Switch"; write_access=0;
coloredValue    DINT    HR1        label = "Colored Value"; color = colorSelection ? color1 : color2 ;
```

The coloredValue color will be always updated according to colorSelection after any change of color1 or color2.

3.1.11.7 Putting it all together. Advanced Expressions Examples

Converting an arbitrary number of seconds to hh:mm:ss format

The following expression shows how to get a string in the form 'hh:mm:ss' from a numeric value containing seconds.In this example *x* contains the total number of seconds to be converted to the desired format.

```
value = [(x/3600).to_s("%02d"), ((x%3600)/60).to_s("%02d"), (x%60).to_s("%02d")].join(":") ;
```

The operators % and / are used to calculate hours, minutes, seconds as numeric values. These are then truncated to integer with the *to_i* method and successively converted to formatted strings with *to_s*. The resulting individual strings are embedded into an array and then joined by means of the the *join* method using ':' as separator.

Given the following two rows:

x DINT HR1 label = "Total Seconds";
t STRING INTERNAL label = "Time"; value =
[(x/3600).to_s("%02d"), ((x%3600)/60).to_s("%02d"), (x%60).to_s("%02d")].join(":") ;

when HR1 holds **3661**, which is 3600 seconds (1 hour) + 60 seconds (1 minute) + 1 second,
the second row will display **01:01:01**

Instead of using the *join* method we could have used the *format* function in a posibly more convenient way. Consider the following:

```
value = format("%02d:%02d:%02d", x/3600, (x%3600)/60, x%60) ;
```

in this case the format specifiers in the format string are just replaced with the relevant time values.

Calculating seconds from a string having the hh:mm:ss format.

Just to illustrate what expressions allow to do let's try now to get the original seconds value from a string already in the hh:mm:ss format. To do so we can use the following expression:

```
value=3600*t.split(":").fetch(-3,0).to_i + 60*t.split(":").fetch(-2,0).to_i + t.split(":").fetch(-1,0).to_i;
```

In this case we extract separately the hours, minutes and seconds as numeric values from the string, we multiply them by 3600, 60 and 1 respectively and then sum them to get the total number of seconds. The extraction of each value from the original string is performed by the *split* method using ':' as delimiter. The relevant element from the split array is obtained with the *fetch* method. We use 0 as the default value for fetching.

Note that we could have used simple array indexing such as `t.split(":")[-3]` to get each part of the original string but this would lead to potential out of bound errors if the original string had some missing part. Particularly, if the original string did only contain minutes and seconds, such as "50:30" (50 minutes, 30 seconds) the referred indexed expression would give an out of bounds error as it would attempt to access a non existing element (the one before the first one). Note also that in all cases we use negative indexing because we interpret that the last part is always meant to be the seconds, the previous to the last one the minutes and so on.

The proposed expression can be optionally optimized by storing the split string in a temporary variable so that the splitting is only performed once. If we apply this optimization.the final solution would look as follows:

tspt STRING[3] INTERNAL value = t.split(":") ;
seconds DINT INTERNAL value = 3600*tspt.fetch(-3,0).to_i + 60*tspt.fetch(-2,0).to_i + tspt.fetch(-1,0).to_i;

Creating a row that alternates between displaying the current time and an arbitrary value

In this example we will create a row that shows a living digital clock showing the current time. Every 5 seconds the time is alternated with a temperature value given in a variable named *temp*. In order to achieve this we enter the following expression in a row.

```
value = $SMPulse10s ? "Time: "+$SMDate.split(" ")[1] : "Temperature: "+temp.to_s("%3.1f")+" °C" ;
```

We use the ternary operator to switch between the time and the temperature depending on the $SMPulse10 system pulse variable. For the clock we take $SMDate and discard the date portion by splitting it out. The temperature is presented formatted with a custom prefix and suffix appended to the actual value.

We can alternatively use the format function to simplify a bit some portions of the expression

```
value = $SMPulse10s ? format("Time: %s", $SMDate.split(" ")[1]) : format("Temp: %3.1f F", 9/5*celsius+32) ;
```

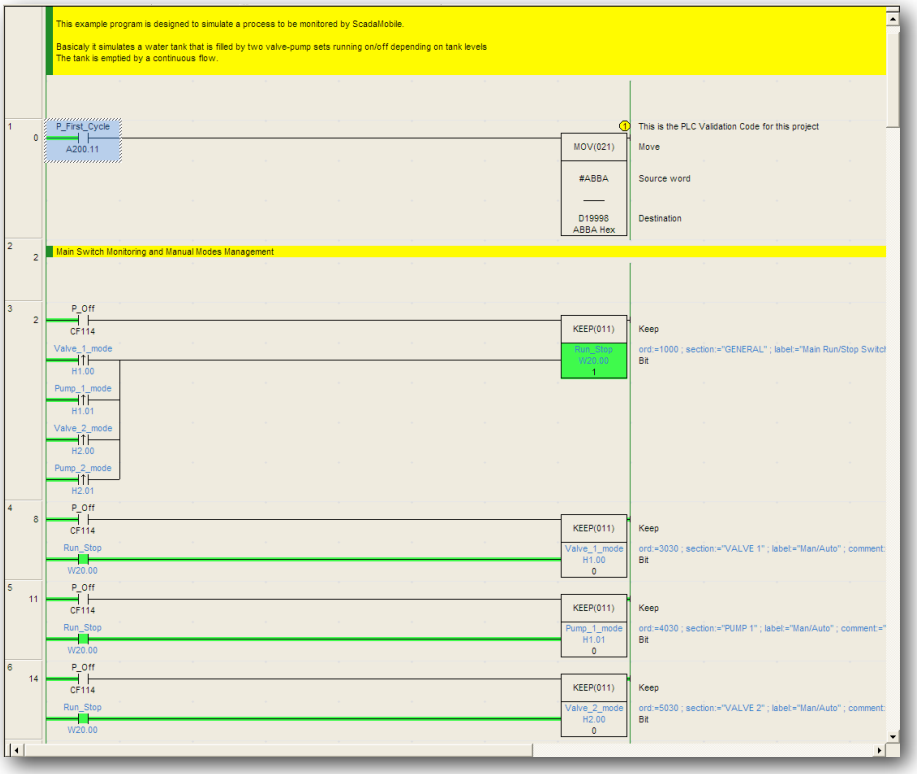
3.2 Omron's CX-Programmer as a Data Source generator.

FINS/TCP support in ScadaMobile includes the possibility to accept files generated directly from Omron's CX-Programmer development environment as Data Sources. The steps to follow in this case are presented next.

3.2.1 Build a project in CX-Programmer.

If you are a PLC systems integrator or programmer you may already know how to use CX-Programmer. In this tutorial we pretend to expose some key steps that you have to know to get your project properly monitored in ScadaMobile. However, a detailed description of how a PLC program is made is beyond this manual. Please refer to the OMRON documentation.

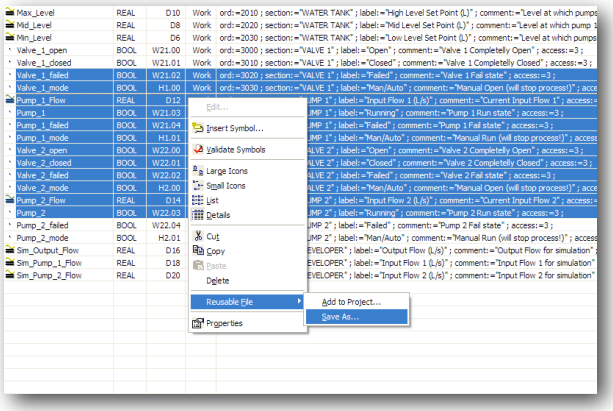
A CX-Programmer project may look like this:



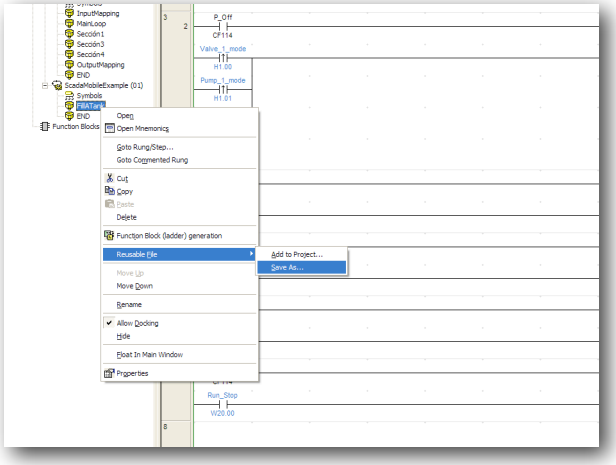
3.2.2 Generate a Source File with variables from CX-programmer.

1. In CX-Programmer select the symbols that you want to monitor in ScadaMobile. Optionally add display attributes for ScadaMobile in the comments column. This is equivalent to filling column D on an Excel generated Source (see Specification of attributes).

2. Export variables in a Reusable symbols file with cxr extension. To do this select 'Reusable File'->'Save As..' on the mouse right click menu over the selected symbols.



3. If you want to perform a bulk export of all symbols used in a particular program section, you can simply export the whole section as a Reusable program file. ScadaMobile will be also able to recognize any symbols in this file.



Files with cxr extension created in this way will be recognized as Sources by ScadaMobile.

3.3 Rockwell RS-Logix 5000 as a Data Source generator

Allen Bradley support in ScadaMobile includes the possibility to accept files generated directly from Rockwell Software RS-Logix development environment as Data Sources. You must follow the following steps.

3.3.1 Build a Project in RS-Logix 5000.

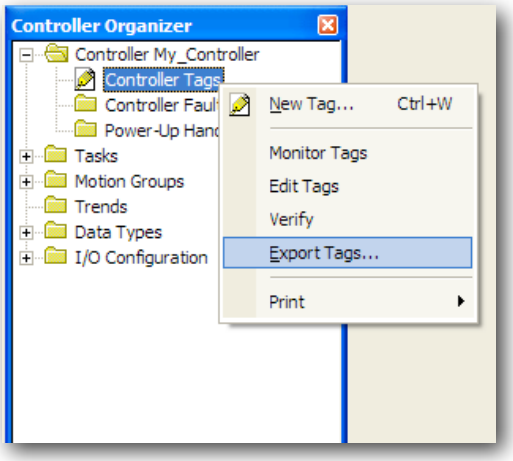
Create or open an already created RS-Logix project. If you are a PLC systems integrator or programmer you already know how to do it. A description on how to program a PLC is beyond this manual. You could need some help from an Allen Bradley integrator or may ask them to do all the steps for you..

3.3.2 Export Controller Tags from RS-Logix 5000.

Unfortunatelly, RS-Logix does not provide a way to select and export selected tags to a file. However it is possible to do a bulk export containing all global Controller Tags:

In RS-Logix, click Export Tags on the right button menu over Controller Tags. RS-Logix will generate a special CSV file containing all global tags present in the project.

Tag Source files generated in this way can contain attributes as described in Specification of attributes. If you need to specify attributes for ScadaMobile you can give them in the Description column of the Controller Tags view in RS-Logix. This is equivalent to filling column D on an Excel generated Source.



NOTE: Files with CSV extension created by RS-Logix as described in this section are structured differently than CSV files created in Excel as described earlier. However, ScadaMobile will recognize them as well.

Since RS-Logix does not provide a convenient way to select a particular set of Tags before exporting, you may need to edit the exported file with Excel or a Text Editor to adapt it to your needs. In such case you do not need to change its overall structure and format, just eliminate redundant rows and you will be done.

3.4 Editing source files in a text editor.

Since Source Files created in any of the ways described above are simply CSV files, they are essentially text files too, so integrators and advanced users can edit them with any text editor such as WordPad or NotePad at their convenience and save them in plain text format. The creation of either CSV or TXT files from scratch in a text editor is also a possibility.

A consideration that must be made if you chose to use a text editor to create or to edit Source Files is that you must be aware of the Double Quoting that Excel applies to text columns containing quotes in it. So you will need to add them manually where necessary. This is not anything particular to ScadaMobile, it is just how CSVs are.

Double Quoting is particularly relevant for column D of a ScadaMobile Source File because many attributes expect a text string as their value, which may be enclosed in quotes. When entering attributes in Excel you do not need to take any special care on this. However, the same file edited in NotePad will reveal the existence of extra quoting that Excel automatically inserts to avoid text conflicts an meet the CSV spec.

As an example, assuming the tag name was main_switch in an Allen Bradley controller, the row referred in section 3.1.4 would look this way on a text editor:

main_switch, BOOL, main_switch, "ord = 1 ; section = ""GENERAL"" ; label = ""Main Run/Stop Switch"" ; comment = ""Main Process Start/Stop"" ; access = 3 ; write_access = 5;"

If you chose to create and edit your files only using a text editor the recommended way is to emulate the TAB SEPARATED CSV format, in which commas are replaced by tabs, which ScadaMobile also supports. Using this format on texts editors is more convenient because it is more readable and you can avoid all the double quoting hassle. The row shown above will look as follows using TABs as instead of commas as separators:

main_switch BOOL main_switch ord = 1; section = "GENERAL"; label = "Main Run/Stop Switch"; comment = "Main Process Start/Stop"; access = 3; write_access = 5;

4 File Import into ScadaMobile.

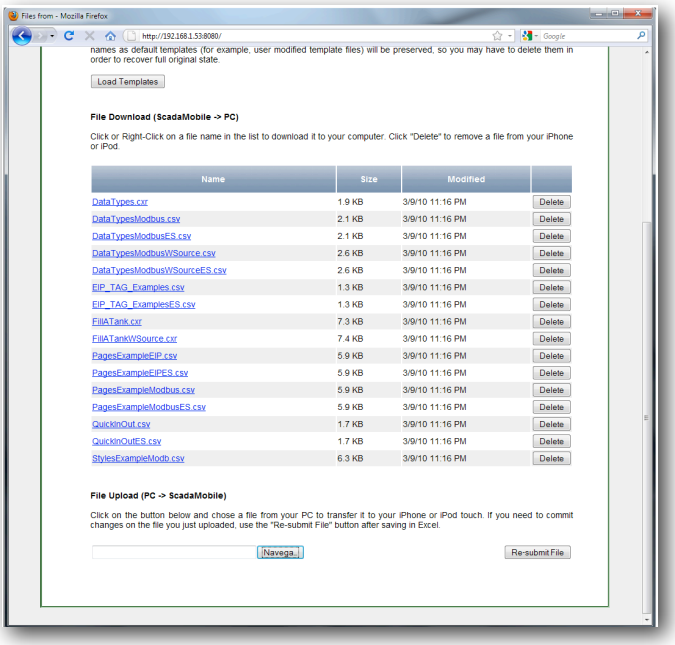
ScadaMobile features an embedded Web server to let users to access to their file contents and to perform various operations. Follow the steps below to use the integrated Web Server

1. Go to 'File Server' tab.

> *NOTE: In order to have access to files you must log into the "administrator" account. The initial password is "admin". ScadaMobile sets itself to this account on first launch. (See User Accounts).*

2. Switch on the File Server.
3. With ScadaMobile's File Server started, copy on your PC's Web browser the IP address provided. If you use Apple's Safari browser you can simply click on the relevant 'Bonjour' link.
4. Now use the available options on the displayed web page in order to move files to/from ScadaMobile. You can upload any files and store them in your iPhone/iPod.

Carrier
5:14 PM
File Server
Files can be transfered between ScadaMobile and a PC using the http link provided below.
Apple's Bonjour technology is supported for Safari browser.
ON
Note: while the server is ON the device will never be put into 'sleep' mode.
http://192.168.1.45:8080
Bonjour: Ordenador de JOAN (6)
Settings
File Server
Files
Connections
Tags


Load Templates
File Download (ScadaMobile -> PC)
Click or Right-Click on a file name in the list to download it to your computer. Click "Delete" to remove a file from your iPhone or iPod.
Name
Size
Modified
DataTypes.cxr 1.9 KB 3/9/10 11:16 PM Delete
DataTypesModbus.csv 2.1 KB 3/9/10 11:16 PM Delete
DataTypesModbusES.csv 2.1 KB 3/9/10 11:16 PM Delete
DataTypesModbusWSource.csv 2.6 KB 3/9/10 11:16 PM Delete
DataTypesModbusWSourceES.csv 2.6 KB 3/9/10 11:16 PM Delete
EIP_TAG_Examples.csv 1.3 KB 3/9/10 11:16 PM Delete
EIP_TAG_ExamplesES.csv 1.3 KB 3/9/10 11:16 PM Delete
FillATank.cxr 7.3 KB 3/9/10 11:16 PM Delete
FillATankWSource.cxr 7.4 KB 3/9/10 11:16 PM Delete
PagesExampleEIP.csv 5.9 KB 3/9/10 11:16 PM Delete
PagesExampleEIPES.csv 5.9 KB 3/9/10 11:16 PM Delete
PagesExampleModbus.csv 5.9 KB 3/9/10 11:16 PM Delete
PagesExampleModbusES.csv 5.9 KB 3/9/10 11:16 PM Delete
QuickInOut.csv 1.7 KB 3/9/10 11:16 PM Delete
QuickInOutES.csv 1.7 KB 3/9/10 11:16 PM Delete
StylesExampleModb.csv 6.3 KB 3/9/10 11:16 PM Delete
File Upload (PC -> ScadaMobile)
Click on the button below and chose a file from your PC to transfer it to your iPhone or iPod touch. If you need to commit changes on the file you just uploaded, use the "Re-submit File" button after saving in Excel.
Navega...
Re-submit File

4.1 Source Files supported by ScadaMobile.

ScadaMobile supports the following source file formats for import:

✓ **csv** - generated by Excel or Open Office

✓ **txt** - any text file with tab, comma or semicolon field delimiters generated by Excel, OpenOffice, NotePad etc.

✓ **csv** - generated by RS-Logix

✓ **cxr** - generated by CX-Programmer

> *NOTE: The recommended way to generate source files is to begin with one of the provided examples and then continue by editing what is missing.*

4.2 Other Files supported by ScadaMobile.

In addition to Source Files, ScadaMobile is able to store and display a number of additional file formats. The following file types (with no aim to be exhaustive) are supported for display:

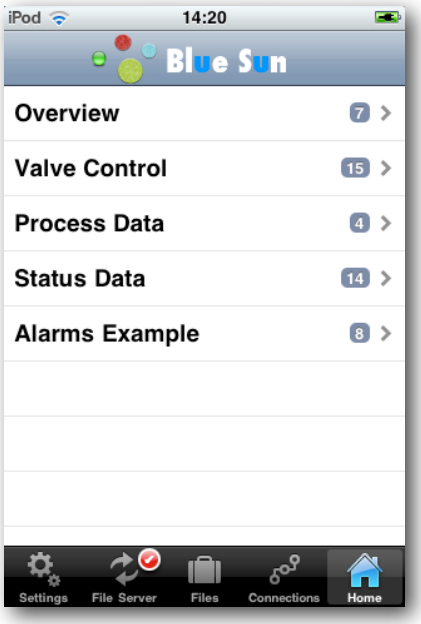
- ✓ **Excel** (.xls)
- ✓ **PDF** (.pdf)
- ✓ **Powerpoint** (.ppt)
- ✓ **Word** (.doc)
- ✓ **Rich Text Format** (.rtf)
- ✓ **Image files** (.png, .jpg, .gif, ...)

The possibility of storing and viewing files can be useful to document specific configurations or simply to move documents between computers.

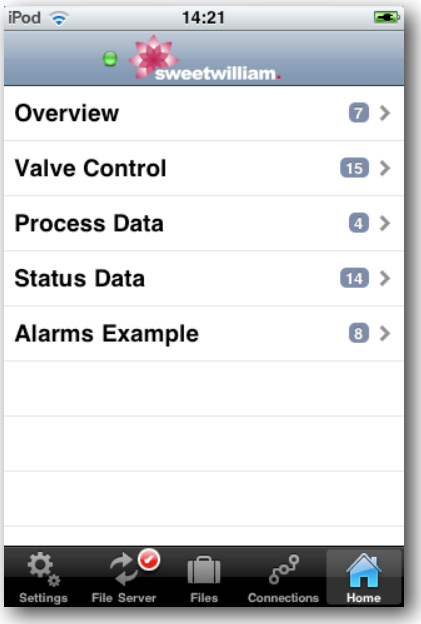
ScadaMobile is able to store any file types even those that are not supported for display.

4.3 Custom Company Logo.

You can place a custom logo on top of the Home view when the list of pages are seen. You do this by using the available options on the Web Page provided by the integrated Web Server. On the left screenshot below an imaginary company logo is shown alongside the same screen with the default logo on the right.



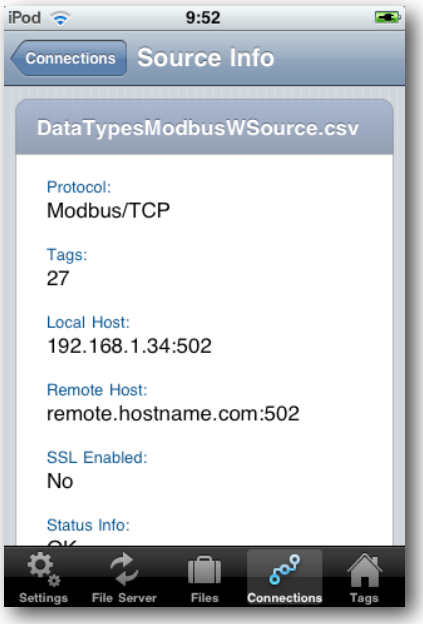
Imaginary Company Logo



Default Logo

5 Connections.

Connections represent links between ScadaMobile and PLCs. Each Connection can have one or more associated Sources. Several Sources are automatically grouped in a single Connection when they share the same communication attributes. You can look at Connection states and their associated Sources in the “Connections” tab view. Also, Sources information, including eventual parsing errors are displayed in it.



6 User Accounts.

User accounts allows to determine what process variables will be visible or editable for a particular use or user depending on particular variable attributes. Relevant attributes for this purpose are 'access' and 'write_access' (See Specification of Attributes)

Only users who have an access level equal to or greater than the one set for a particular variable will be able to view or edit it.

The purpose of user accounts is to make possible that a PLC programs developer or integrator would create a pre-configured solution for a particular final customer. A final customer does not usually need to change communication settings or to modify attributes. This is normally a developer's responsibility. Therefore, by setting user accounts and leaving the 'administrator' password private, a developer can choose to provide customized, safe access to a process being monitored.

User accounts can also help to prevent non-authorized persons from interacting with remote processes by getting physical access to an iPhone with ScadaMobile installed.

Two accounts are set up by default with the following names, passwords and access levels:

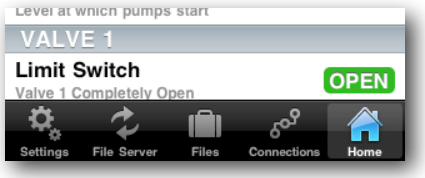
USER	PASSWORD	ACCESS
administrator	admin	9
nobody		0

The "administrator" account lets you create additional accounts with intermediate access levels as well as to select active accounts.

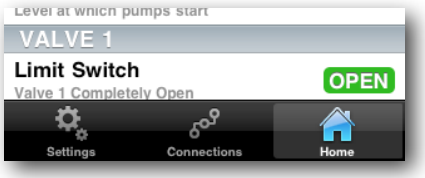
It is strongly recommended that you change the default passwords once you have a running configuration in place. Note that there is no way to retrieve a lost password. The only way to recreate the original passwords is by removing ScadaMobile from your iPhone and iTunes and reinstalling from the App Store. If you ever need to do so, be sure your Source Files are conveniently backed up.

Restrictions for Non-Administrator users

Only the "administrator" account has full access to all the application Non-administrator accounts have restricted access to deployment features and not all tabs are available to them. The following screenshots show the available tabs for the Administrator user (left) and for a regular user (right)



administrator user



regular user

Managing accounts

Accounts are managed in a navigation interface similar to the Contacts application The screenshot below shows that a new account named “Tommy” has been created, which has an access level of 3 and that it is active.

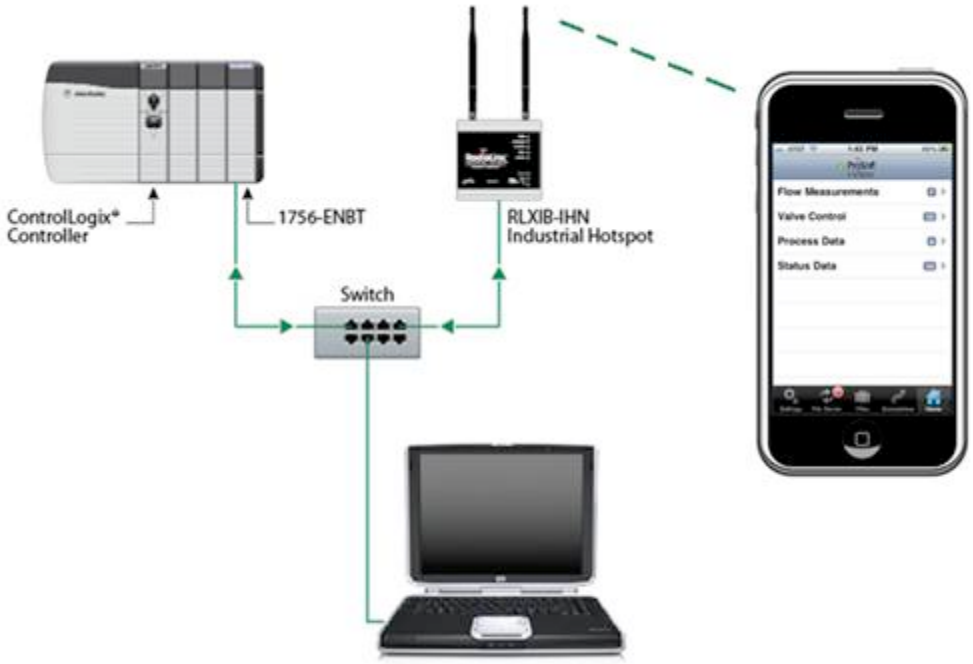
In order to log in a particular user tap on ‘Current account’.

You can also make ScadaMobile to ask for the current account password at launch. In order to do so set the ‘Automatic login’ switch in ‘Settings’ to ‘Off’.

7 Network Settings for local access.

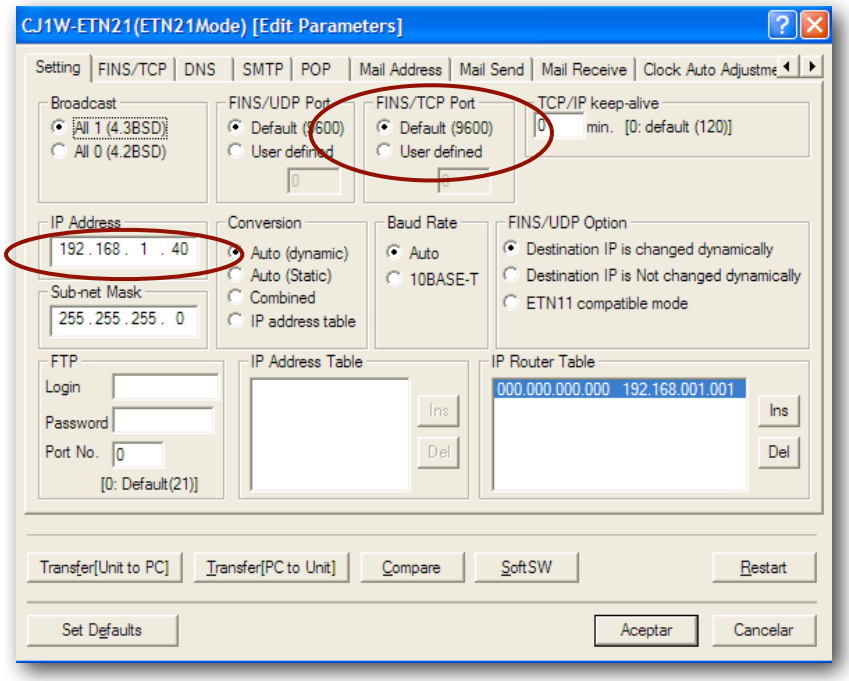
ScadaMobile uses wireless TCP/IP technology to connect and to communicate with PLCs. Direct access from a local network requires both devices to be in the same subnet. The PLC acts as the communications server and the iPhone or iPod Touch is the client.

The following picture shows a typical setup using the recommended industrial wireless hardware, but basically any WiFi router will do it.



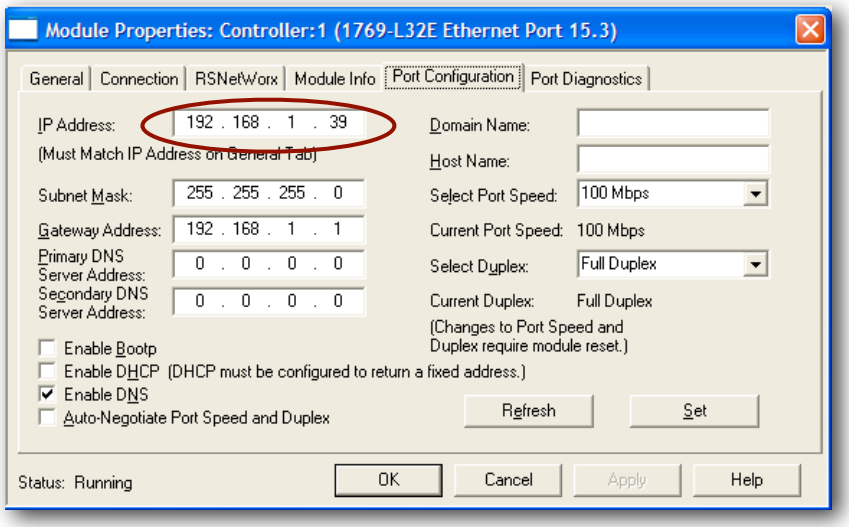
7.1 PLC Settings for local access.

In case of Omron's **Fins/TCP** protocol use CX-Programmer tool to set a fixed local IP and Port for the PLC on the ethernet configuration panel.



For **EIP/Native** protocol and Allen Bradley controllers use RS-Logix 5000 tool to set a fixed local IP for the PLC on the ethernet module properties panel.

For **EIP/PCCC** protocol use Allen Bradley's RS-Logix 500 tool and set a fixed local IP for the PLC on the Channel Configuration panel

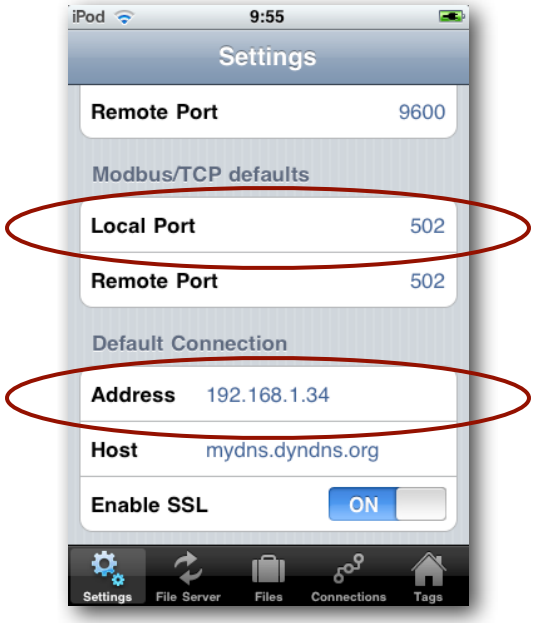


For PLCs or devices based on the **Modbus/TCP** protocol or **Siemens/ISO_TCP** consult the relevant vendor documentation to know how to set ports and addresses.

7.2 ScadaMobile Settings for local PLC access.

ScadaMobile configuration depends on whether you have specified communication attributes for Source Files. See Specification of Attributes for further information.

For Sources without communication attributes ScadaMobile uses the default ports and addresses entered into the relevant fields on the “settings” tab view. Write the values you used for the PLC settings.



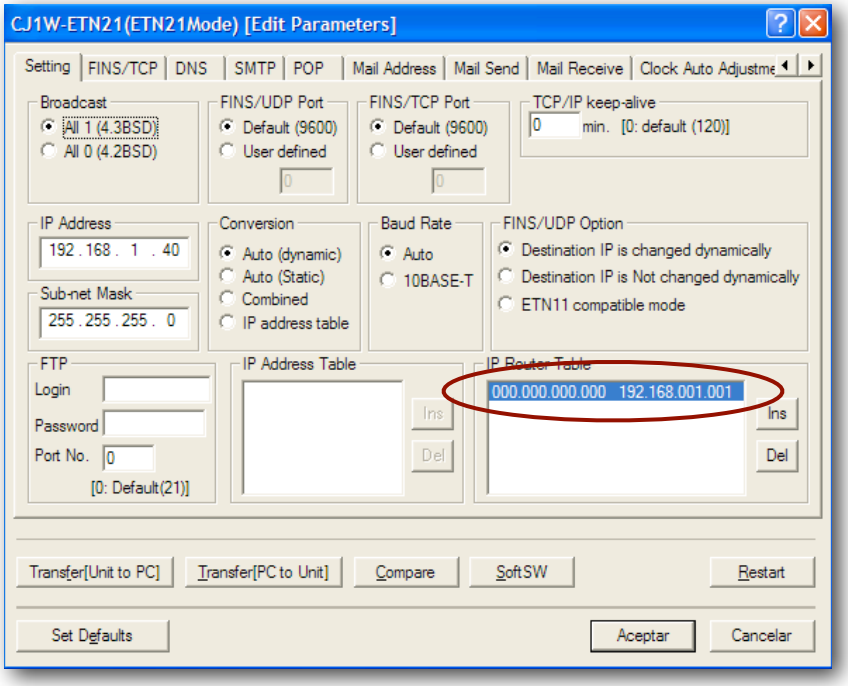
8 Network Settings for remote access.

ScadaMobile is designed to communicate with PLCs without using dedicated servers or any specific software installed on a PC. ScadaMobile communicates with PLCs by using industrial protocol commands.

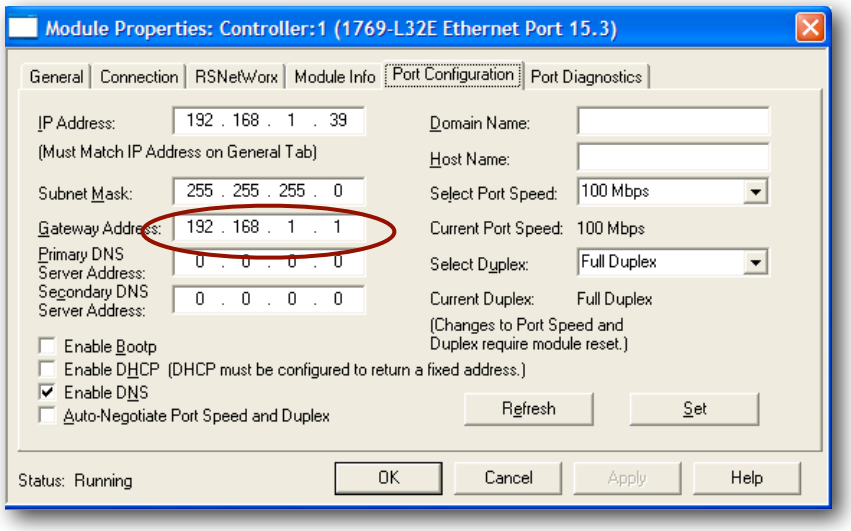
To establish a remote connection, a GPRS or DSL router is needed at the PLC site, which will act as a bridge between the LAN (Local Network) where the PLC is installed and the WWAN or WAN (Internet) to which a remote iPhone or iPod Touch will have access to. This figure shows a standard setup.

First determine the LOCAL IP address of the GPRS or ADSL router. PLCs need to know the router address as it is the gateway to the internet.

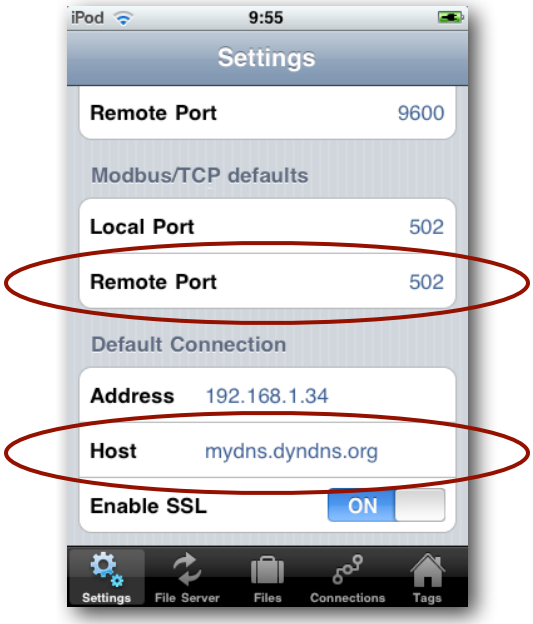
1. In case of **Omron Fins/TCP** copy the router address in the 'IP Route Table' field of the ethernet configuration panel for the PLC in CX-Programmer.



2. For **EIP/Native** protocol and Allen Bradley controllers use RS-Logix tool to set the fixed local router IP (gateway) on the ethernet module properties panel.
3. For **EIP/PCCC** protocol use Allen Bradley's RS-Logix 500 tool and set the gateway IP on the Channel Configuration panel


Module Properties: Controller:1 (1769-L32E Ethernet Port 15.3)
General
Connection
RSNetWorx
Module Info
Port Configuration
Port Diagnostics
IP Address: 192 . 168 . 1 . 39
(Must Match IP Address on General Tab)
Subnet Mask: 255 . 255 . 255 . 0
Gateway Address: 192 . 168 . 1 . 1
Primary DNS Server Address: 0 . 0 . 0 . 0
Secondary DNS Server Address: 0 . 0 . 0 . 0
Domain Name:
Host Name:
Select Port Speed: 100 Mbps
Current Port Speed: 100 Mbps
Select Duplex: Full Duplex
Current Duplex: Full Duplex
(Changes to Port Speed and Duplex require module reset.)
Enable Bootp
Enable DHCP (DHCP must be configured to return a fixed address.)
Enable DNS
Auto-Negotiate Port Speed and Duplex
Refresh
Set
Status: Running
OK
Cancel
Apply
Help

4. For **Modbus/TCP** based devices or **Siemens S7** controllers refer to the vendor's documentation.
5. Now log into the GPRS or DSL Router and configure NAT options to set up a bridge between the WAN and your PLC local address and port. Note that the default port number is 44818 for Ethetnet/IP, 502 for Modbus/TCP, and 9600 for Omron PLCs. Protocol on the router must be set to TCP/IP. Look at your router documentation for details.
6. If you have a fixed IP address enter it as such in ScadaMobile either as a Settings default or embedded in Source Files. .
7. If your router access the WAN through a dynamic IP then you must create an account with a dynamic DNS services provider such as www.dyndns.org, and configure your router to notify of IP changes. In this case, enter in ScadaMobile the name you chose for your dynamic DNS. The port number must still be the one configured in the NAT section of your router.



9 Security.

ScadaMobile networking security is based on TCP/IP technology and depends in part on the security features available in the router installed at the PLC location.

For local connections through WiFi security is given by the wireless network security protocol in use. WPA and WPA2 with a strong password is the recommended security protocol.

For remote connections, an iPhone or iPod touch is able to make use of secure data tunnels by enabling VPN. If your router supports L2TP/IPSEC or PPTP then you will be able to create this kind of connection. Most medium to high-end DSL or Cable routers support at least PPTP. VPNs client connections are configured on the iPhone with the General Settings App.

Some routers can be loaded with a SSL certificate and be configured to bridge incoming SSL requests from the WAN to unencrypted TCP on the LAN side. ScadaMobile supports TLS-SSL encryption. You can activate TLS-SSL in ScadaMobile to provide communications confidentiality if your router supports SSL/TCP bridging.

For most protocols, ScadaMobile provides an independent way to protect users from undesired access by persons using uncontrolled ScadaMobile copies. This is done by setting a Validation Code both in the PLCs and ScadaMobile which will prevent ScadaMobile to access PLCs unless both codes match. Next section describes validation codes and how you can set them.up.

Finally, physical access can compromise security. It is relatively easy for an unauthorized user to gather physical access to a device and run a remote monitoring application. To fight this possibility, ScadaMobile's user accounts provide password based security. By setting off the 'automatic login' switch in ScadaMobile settings tab, a password key will be asked each time the app is launched, thus preventing people not knowing the password from using the app. Additionally Apple provides a service for blocking lost or stolen devices so that no one is able to access to data or execute apps in them until the real owner reactivates them.

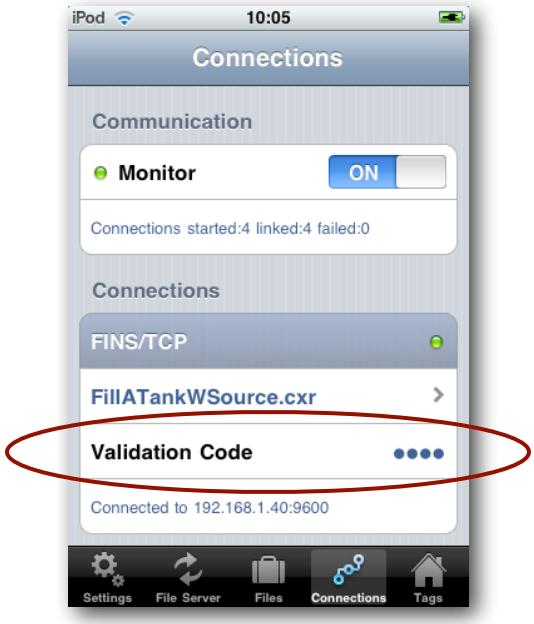
9.1 Validation Codes.

For most protocols ScadaMobile requires a validation code being held by the PLC, which is queried on each connection. This password must be stored in your PLC as a 16 bit hexadecimal value (0 to FFFF) and must match the value specified in 'Validation Code' for connections to that PLC to succeed. In most cases this security measure alone is enough for simple applications.

Validation Codes are stored in PLCs in the following Memory Address or Tag depending on protocol.

PROTOCOL	DEFAULT VALIDATION TAG	REMARKS
EIP/Native	SMValidationTag	Any INT value. This tag **must be present** in order for ScadaMobile to communicate. Set initially to '0' to avoid having to enter it on ScadaMobile during development stages.
EIP/PCCC	N98:0	Any INT value. This tag **must be present** in order for ScadaMobile to communicate. You may have to create a Data File number 98 of type Integer with at least 1 element
Fins/TCP (Omron)	D19998	Any value from #0000 to #FFFF is valid.
Modbus/TCP **Modbus over TCP**	(Not Available)	See note below.
Siemens/ISO_TCP	MW998	Any value from #0000 to #FFFF.
The Validation Code feature is not available for Modbus/TCP due to the great number of Industrial devices supporting this protocol, which makes impractical to establish a general way to implement such feature.		

Validation codes are entered in the relevant fields of ScadaMobile' Connections View. ScadaMobile will store and remember codes between connections but it will reset them to '0' for connections that changed as a result of sources change.



Note that ScadaMobile will always perform this security check. There is no way to disable or prevent it, however you can set a custom *validation tag*:

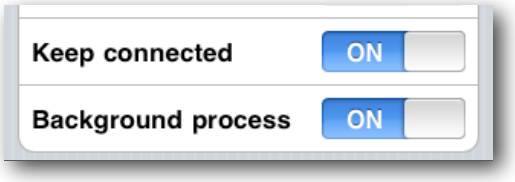
Custom Validation Tag.

If the default validation tag interferes with your project you can set a custom one with the *validation_tag* attribute. When you use custom validation tags you must follow the following points.

- Be sure to have the same validation tag in all files sharing a single connection, (i.e with identical communication settings), otherwise one or more files will appear as dissociated on the Connections view.
- If you explicitly set the *validation_tag* attribute to the default one, you will still have to do the same in all your files sharing a connection.
- When you explicitly set a validation tag, you will have to explicitly set a non zero value for the validation code, as using 0 as validation code is explicitly forbidden in this case and doing so will always fail the validation check.

10 Background Task Processing

You can tell ScadaMobile to keep connections alive even when the device (iPhone, iPod) is locked. Additionally, ScadaMobile supports running in the background during a finite period of time on multitasking enabled devices. On the settings tab there are two switches that allow for enabling such options.



Keep connected.

When ON, ScadaMobile will keep any open connection as is and will continue polling PLCs even when the device is locked. This is intended to help keeping alarms and graphs updated, but it may have a negative impact on battery life.

When OFF, ScadaMobile will close any open connection when another application becomes active or the device goes to the lock state, either by the Auto-Lock timeout or by pushing the bottom on the edge. This will prevent excessive battery drain and may reduce carrier fees in case of cellular remote connections. When the device is unlocked and ScadaMobile becomes active again, communications with PLCs will be restarted.

Background process.

When ON, the app will continue running in the background for a finite period of time while other applications are active, thus allowing for keeping alarms and graphs updated and enabling local alarm notifications while the user is performing other tasks.

When OFF, the app will not explicitly start a procedure to run in the background, and local notifications will be disabled. However, given how the iPhone iOS behaves, the app may still be running in the background for an undetermined period of time if enough resources are available or the user does not explicitly switch to a different app.

The maximum time applications are allowed to run in the background or in the lock state is an Apple choice and it is currently set at 10 minutes. Therefore ScadaMobile, just as any other application, will be switched off or suspended after this time has expired. Therefore, end users need to be instructed to put ScadaMobile in the foreground as soon as they have completed other tasks and not to allow the device to go to sleep, if keeping alarms and graphs updated at any time is essential.

11 Performance

ScadaMobile is designed to offer great performance and a satisfactory user experience with good interface responsiveness. Integrators do not usually need to take particular actions to improve performance. The app carries out a series of optimizations on tags and PLC communications that in most cases are enough to relieve integrators from having to adopt particular actions or design patters.

Still, understanding what optimizations ScadaMobile performs can be useful to find out why, if ever, a performance issue arose and what you can do to improve on it.

Performance matters tend to be a complex subject and extensive discussions can be made; ultimately, testing is the only way to have a trustable answer. We will just enumerate some of the techniques used in ScadaMobile to maximize performance and will give some tips to help you getting the best of the app for your particular project.

ScadaMobile automatically performs the following optimizations:

- Only the minimum set of tags necessary to display the interface is polled at a given time. Particularly, tags that are not used in expressions and which values are not shown on the interface at a given time are not polled at all.

  ➡ *Therefore, the less tags you link with expressions or use in expressions the better. In fact, having a large number of tags for the single purpose of showing their PLC value will not incur in any performance penalty because most of the time they will not be polled out from the PLC anyway.*

- If a particular PLC address needs to be read at a given time, it will only be read once per polling cycle regardless of how many instances of tags tied to that address are found in the project.

  ➡ *Consequently, you can repeat the same PLC tag address as many times as you need in your project without having to worry about any performance penalty.*

- The list of candidate tags to be read at a given time is applied a minimal cost algorithm to determine the less and shorter available communication frames for the relevant protocol which fulfill the entire request. The optimal solution may include reading unsolicited tags if this justify a better performance figure. The simpler scenario is reading several, almost contiguous tags, with a small gap in between. Reading all of them and then discarding the unused ones may be faster than only reading the solicited ones.

  ➡ *To help on this, you can use tags as contiguous as possible and leave the shorter possible gaps between them, however note that if you have spared tags that are used in expressions, or your users make extensive use of the plotting feature, this measure may not be as effective as you might expect or even unproductive, because basically you may not always beat the already optimal set of commands that ScadaMobile would generate anyway.*

- Tags used in alarms or used in expressions are permanently polled to guarantee interface consistency. These tags are also candidates to automatic optimization and applied all the above techniques, but since they are read more often they offer the best opportunity for integrators to optimize their project in an user noticeable way.

  ➡ *As said, this is by far the best thing you can do to optimize your project. The key is to have as few as possible tags tied to expressions and to keep them in contiguous PLC memory addresses, specially with no intercalations of tags that are not used in expressions (i.e. tags that are candidates to be stripped off from readings). In other words, keep the tags used in expressions or alarms as packed as possible in PLC memory and do not interlace them with the remaining tags.*

  ➡ *If you use a lot of boolean tags consider boolean arrays on EIP/Native or bit access on registers on other protocols. Bit readings on registers are faster because you get several boolean values with a single register read.*

Finally, ScadaMobile communications are completely moved to secondary threads and executed asynchronously in order to minimize any impact to the user interface execution. This essentially decouples user interface execution from PLC communications. From a user perspective, the app behaves as if only the user interface was running.

If you are concerned about performance or you are planning a really big project, the best advice is to do some planning following the above recommendations even before implementing your project. Then profile periodically for performance as you add more rows to it.

12 Examples

The ScadaMobile application comes with pre-installed examples that can be used directly or can be downloaded to a PC to be used as templates for your own development. The provided examples are the following:

EXAMPLE	CREATED WITH	REQUIRED PROTOCOL	COMMUNI-CATION ATTRS.	STYLES	ALARMS	EX-PRES-SIONS
QuickInOutES.csv	MS Excel	Fins/TCP	NO	NO	NO	NO
DataTypes.cxr	CX-P	Fins/TCP	NO	YES	NO	NO
FillATank.cxr	CX-P	Fins/TCP	NO	NO	NO	NO
FillATankWSource.cxr	CX-P	Fins/TCP	YES	NO	NO	NO
DataTypesModbus.csv	MS Excel	Modbus/TCP	NO	YES	NO	NO
DataTypesModbusWSource.csv	MS Excel	Modbus/TCP	YES	YES	NO	NO
EIP_TAG_Examples.csv	MS Excel	EIP/Native	NO	NO	NO	NO
EIP_PCCC_Examples.csv	MS Excel	EIP/PCCC	NO	NO	NO	NO
PagesExampleModbus.csv	MS Excel	Modbus/TCP	NO	NO	NO	NO
StylesExampleModb.csv	MS Excel	Modbus/TCP	NO	YES	NO	NO
StylesExampleEIP_PCCC.csv	MS Excel	EIP/PCCC	NO	YES	NO	NO
AlarmsModbus.csv	MS-Excel	Modbus/TCP	NO	YES	YES	NO
AlarmsEIP_PCCC.csv	MS Excel	EIP/PCCC	NO	YES	YES	NO
ColorfulControlsModbus.csv	MS-Excel	Modbus/TCP	NO	YES	NO	NO
ColorfulControlsEIP_PCCC.csv	MS Excel	EIP/PCCC	NO	YES	NO	NO
CommunicationsState.csv	MS Excel	any	NO	YES	NO	YES
Formula-ONE.csv	MS Excel	none	NO	YES	YES	YES
Aphorism-ONE.csv	MS Excel	none	NO	YES	NO	YES
International-ONE.txt	Text Editor	none	NO	YES	NO	YES

QuickInOut.csv

This example displays Inputs in channel 0 and Outputs in channel 1 for supported Omron PLCs. No special settings or program installed in the PLC are needed. This file was created in Excel.

DataTypes.cxr

Similar to the previous one but acting on internal PLC areas. This serves the purpose of showing the supported data types. It can be imported directly into a CX-Programmer's symbol table. This file was created in CX-Programmer, but does not require any program in the PLC.

FillATank.cxr

This is the most comprehensive example for Omron PLCs and it consists of a simulation of a water tank that is continuously emptied, and filled by means of two Valve and Pump pairs with configurable levels and flows.

The file was created in CX-Programmer and includes a complete program section that can be imported and run in any supported Omron PLC. Since the process is actually simulated and run by a PLC it should be transfered to a PLC in order to get meaningful monitoring on ScadaMobile.

This example also features several access levels for the monitored process variables, so you will also be able see how variables display change depending on current user's access level.

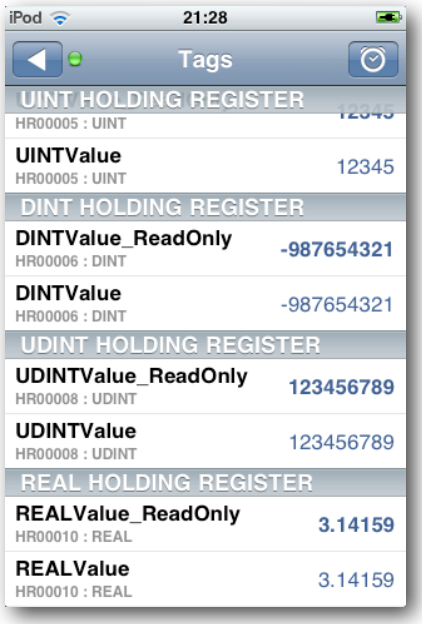
NOTE: The provided PLC program section uses TIMX instructions, so you may have to select 'Execute timer/counter as binary' on the Properties settings of the PLC before importing this example into CX-Programming.

FillATankWSource.cxr

Identical to the previous example except that communication attributes are incorporated into the file.

DataTypesModbus.csv

This serves the purpose of showing the supported data types with a focus on Modbus devices. This file was created in MS Excel and does not require any program in the PLC.

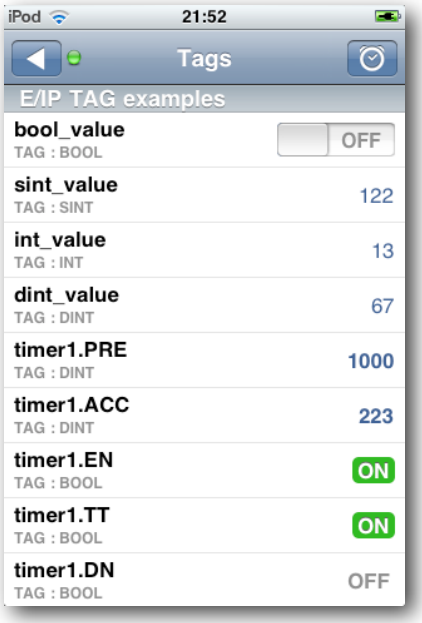


DataTypesModbusWSource.csv

Equal to previous example but containing communication attributes.

EIP_TAG_Examples.csv

The purpose of this example is to demonstrate a set of valid symbolic tag names while accessing Allen Bradley Logix controllers.

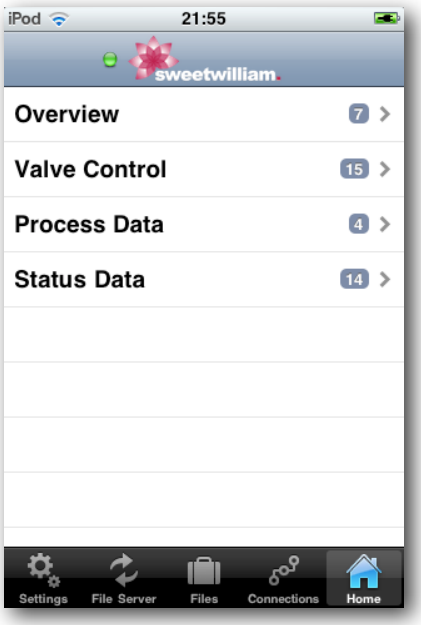


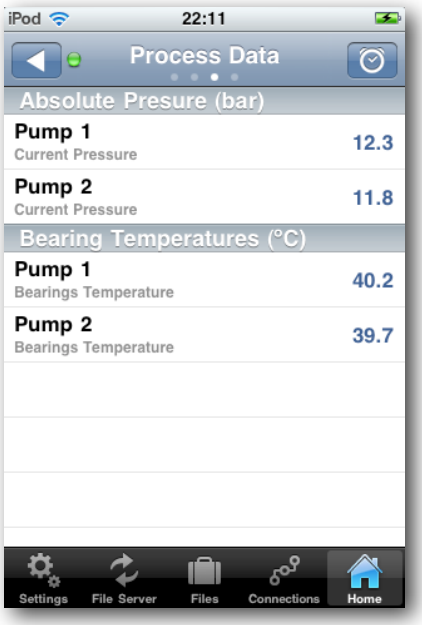
EIP_PCCC_Examples.csv

The purpose of this example is to demonstrate a set of valid symbolic tag names while accessing Allen Bradley Micrologix controllers.

PagesExampleModbus.csv

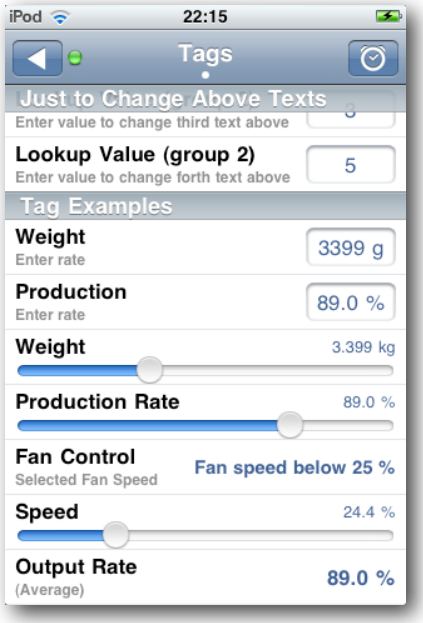
This example shows how to arrange tags in several pages. It uses arbitrary Modbus Coils and Registers.

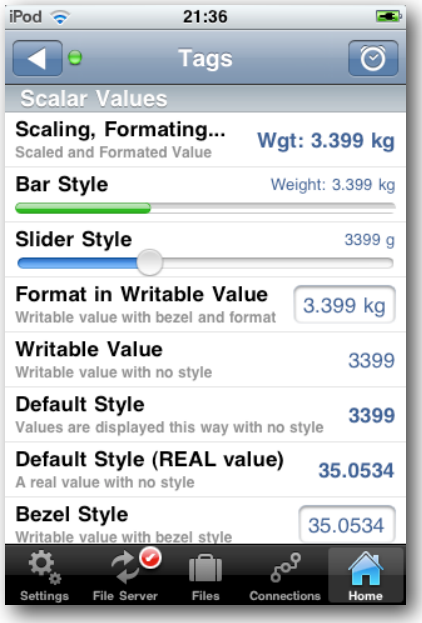


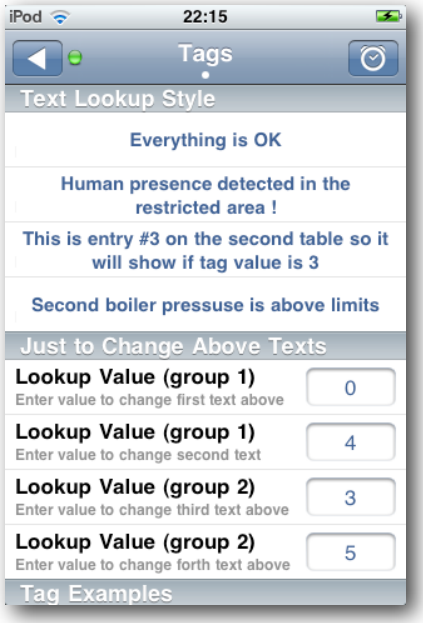


StylesExampleModb.csv

This example shows the available styles and some related tag attributes such as scale, format, bounds, prefix, suffix as well as the use of lookup tables.





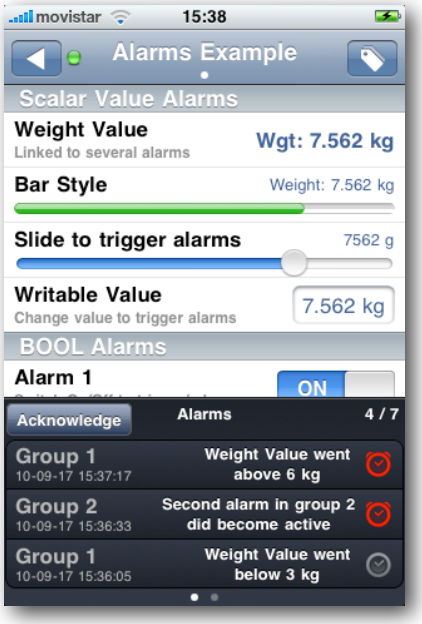

iPod 22:15
Tags
Text Lookup Style
Everything is OK
Human presence detected in the restricted area !
This is entry #3 on the second table so it will show if tag value is 3
Second boiler pressuse is above limits
Just to Change Above Texts
Lookup Value (group 1)
Enter value to change first text above
0
Lookup Value (group 1)
Enter value to change second text
4
Lookup Value (group 2)
Enter value to change third text above
3
Lookup Value (group 2)
Enter value to change forth text above
5
Tag Examples

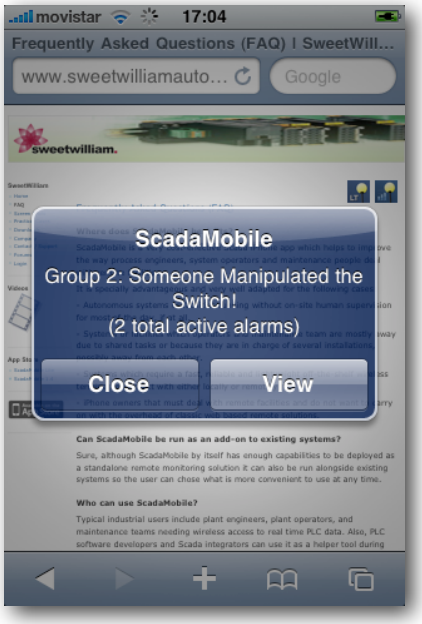
StylesExampleEIP_PCCC.csv

This example shows the available styles and some related tag attributes such as scale, format, bounds, prefix, suffix as well as the use of lookup tables using EIP/PCCC protocol and will run on a AB Micrologix Controller.

AlarmsModbus.csv

Demonstrates Alarms management on a Modbus configuration. The example includes both alarms based on tag value and discrete alarms. The following screenshots show alarming in the application (left picture) and an alarm notification from ScadaMobile while running in the background, which happened while browsing the internet in Safari.



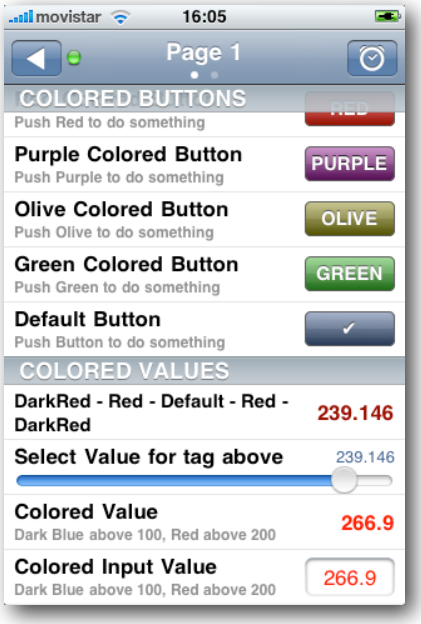


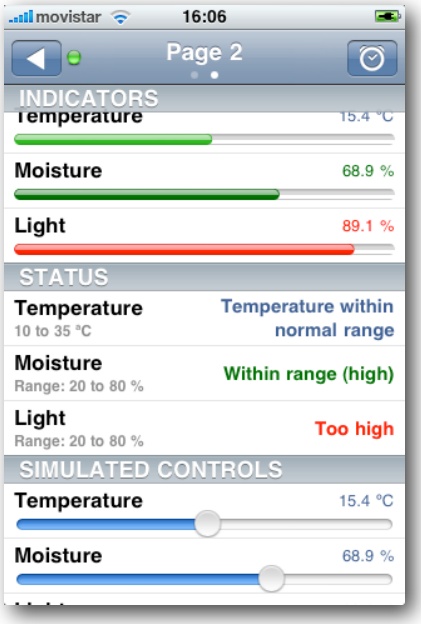
AlarmsEIP_PCCC.csv

Identical to the previous one but based on the Ethernet/IP PCCC protocol.

ColorfulControlsModbus.csv

Presents ways to specify colors based on value, in tags and controls.The example is based on the Modbus/TCP protocol.





ColorfulControlsEIP_PCCC.csv

Identical to the previous one but based on the Ethernet/IP PCCC protocol.

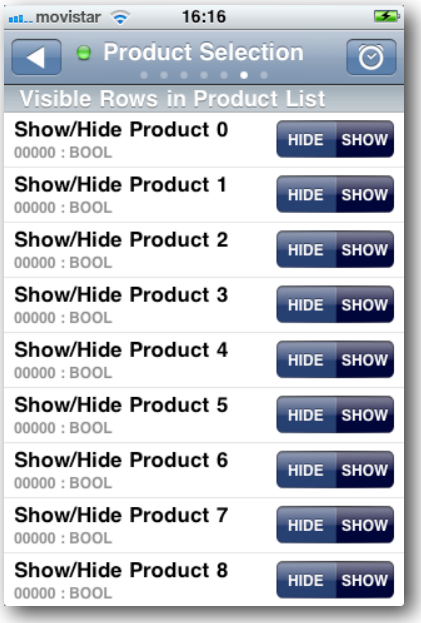
CommunicationsState.csv

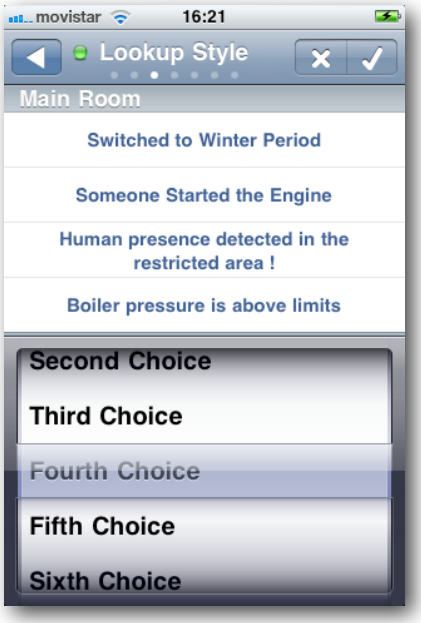
Uses the system communication state global variable and displays a row showing the current communications state.

Formula-ONE.csv

Shows several aspects and possibilities of using expressions on attributes. The example is self contained by using only INTERNAL tags and it does not need an actual PLC connection to run. It is structured in pages and fully commented to demonstrate:

- Switching and displaying/hiding of rows on the interface based on user selectable style.
- Use of the 'picker' attribute to allow users to retrieve a value based on a picker wheel control.
- Complete or partial removal of rows, sections and pages based on simple boolean states.
- Arithmetic and boolean calculations based on tag values
- Implementation of controls to enable/disable particular alarms
- Use of system variables in expressions to achieve special effects.

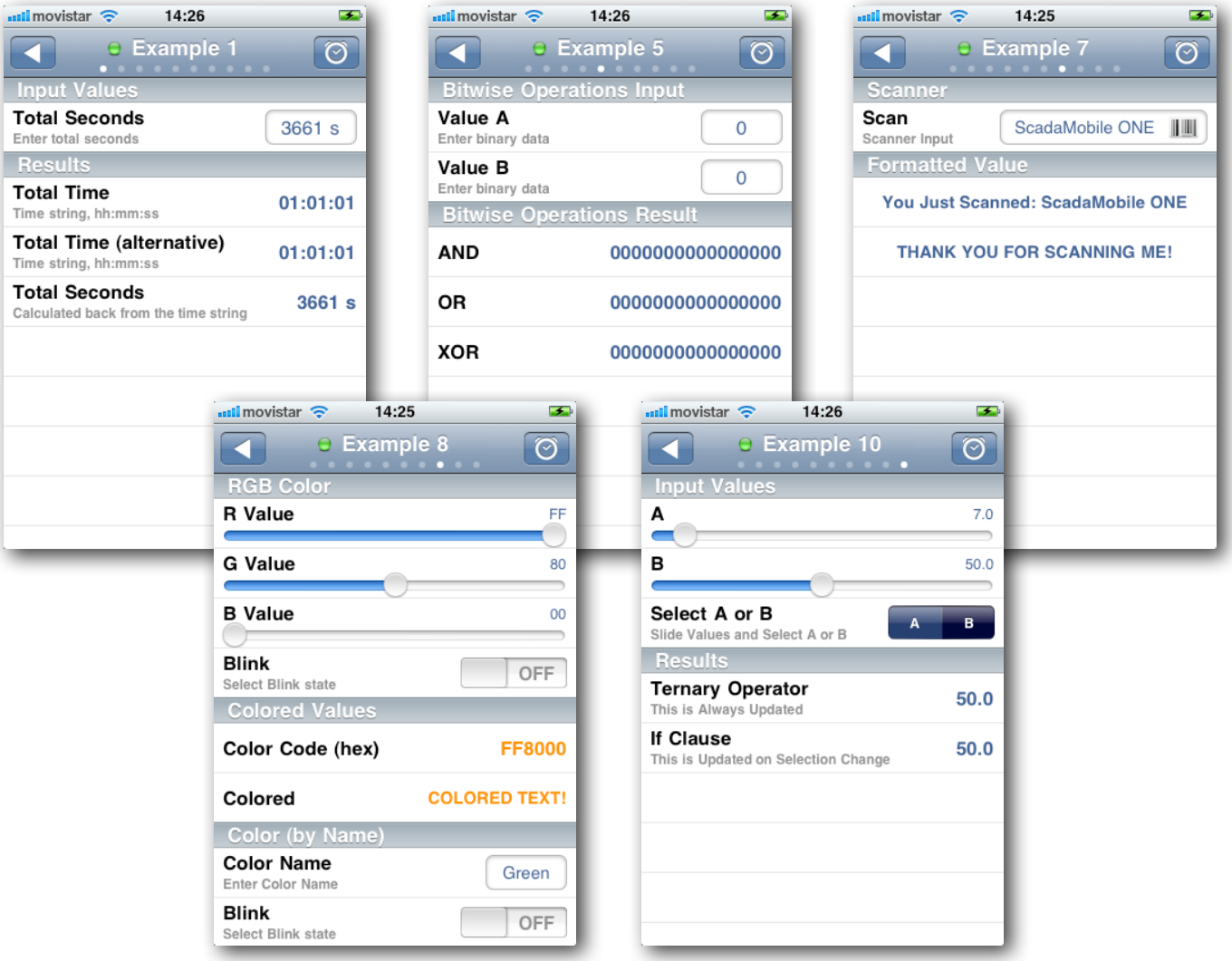




Aphorism-ONE.csv

Demonstrates de use of advanced Numeric, String and Array expressions on attributes. A feature is shown on each page approach. The example is self contained and runs using only INTERNAL tags. The following goals are demonstrated:

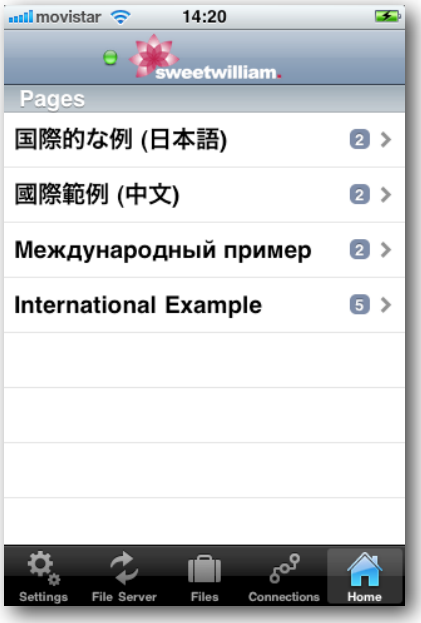
1. Convert total seconds to a string in the form hh:mm:ss
2. Alternate display Time/Temperature. Conversion between degrees Celsius and Fahrenheit.
3. Display Date/Time string in several formats: Default, American Style, German Style
4. Convert to and from Rectangular and Polar coordinates.
5. Use of Binary Input and demonstration of Bitwise Operations.
6. Access to the global Lookup Table and use in formatted expressions.
7. Demonstration of the Bar Code Scanner tag style.
8. Arbitrary custom Coloring and Blinking of values.
9. Arbitrary custom Text on Boolean values
10. Conditional operations using either the Ternary operator or the If Clause



International-ONE.txt

Demonstrates the use of International Languages in Text attributes and String expressions.

This file was written in a text editor and saved as Unicode UTF-16 text format. The example is self contained and works using only INTERNAL tags.



Document Revision History

Refer to this section to look at changes on this document over different versions.

Version 2.1.0.

- Replaced occurrences of 'Tag' by 'Variable' where referring to SM Variables as opposed to PLC tags.
- Added section: "Representation of Character Strings in PLCs".
- Added Siemens/ISO_TCP in the list of supported protocols and references to it where relelvant.
- Updated description of the 'value' tag attribute in "Tag Attributes" section.
- Added sub section: "Global Attributes for Siemens S7 Controllers" under "Global Attributes" heading.
- Updated sub section: "Multiple range lookup tables" under "Look-up Tables" heading.
- Changes in section: "Data types in Expressions".
- Updated section: "Performance".
- Removed Example: "PagesExampleEIP.csv"
- Added Example: "CommunicationsState.csv"
- Added Example: "International-ONE.txt"
- Added screen shots for Example: "Aphorism-ONE.csv"

Version 2.0.0.

- New styled table of contents.
- Added section: "Tag Scope".
- Added memory arrays definition in section "Specification of Variable Types".
- Mention of how to specify communication protocol in section "Specification of Variable Addreses".
- Incorporated references to the new strings type in section "Specification of Variable Addreses".
- Added reference for accessing program tags in Logix controllers in section "Specification of Variable Addreses".
- Added section: "Internal Tags".
- Replaced all previous references to Local tags by "Internal tags".
- Replaced all previous references to arrays by "value lists" and text strings by "text"
- Added section: "PLC Memory Arrays".
- New 'bool', 'barcode' and 'validation_tag' attributes or styles added to the "Tag Attributes" section.
- Changed placement of sections: "Comments in Data Sources" and "Specification of Communication Protocol".
- Extended section "Expressions" to cover Strings and Arrays.
- Extended and renamed section: "Supported Operators and Operator Precedence".
- Added section "Data types in Expressions", extending and replacing some subheadings in the old "Expressions" section.
- Updated section "System Variables".
- Added section "Functions Methods and more about Operators".
- Added section "Putting it all together. Advanced Expressions Examples".
- Added chapter "Performance".
- Added "Document Revision History".

SweetWilliam, S.L. Contact Information

SweetWilliam, S.L.
Llevant, 10
17844 - Cornellà del Terri - Spain
Tel: +34 972 59 51 39

e-mail: support@sweetwilliamsl.com
Web: http://www.sweetwilliamsl.com